Rejection-Free and Partial Neighbor Search MCMC Algorithms

by

Sigeng Chen

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy

Department of Statistical Science
University of Toronto

Rejection-Free and Partial Neighbor Search MCMC Algorithms

Sigeng Chen
Doctor of Philosophy

Department of Statistical Science
University of Toronto
2023

Abstract

The Metropolis algorithm [39], [59] involves producing a Markov chain to converge in distribution to a specified target density π. To improve its efficiency, we can use the Rejection-Free version of the Metropolis algorithm, which avoids the inefficiency of rejections by evaluating all neighbors. Rejection-Free can be made more efficient through parallel hardware. However, for some specialized hardware, such as Digital Annealing Unit [57], the number of neighbors being considered at each step is limited. Hence, we propose an enhanced version of Rejection-Free known as Partial Neighbor Search, which only considers a portion of the neighbors. Partial Neighbor Search can be applied efficiently despite the number of neighbors. Especially for continuous cases with uncountable many neighbors, Partial Neighbor Search can be applied easily and samples efficiently while Rejection-Free is not feasible, and the Metropolis algorithm is slow. Both algorithms can be used in many other circumstances as well, such as the optimization question. In combinatorial optimization, Simulated Annealing using Metropolis steps at decreasing temperatures are widely used to solve complex problems [47]. In order to improve its efficiency, we can also use the Rejection-Free version of the Metropolis algorithm, which avoids the inefficiency of rejections by considering all the neighbors at every step. In addition, in optimization questions, Partial Neighbor Search can not only be helpful when being applied on parallel hardware, but it can also avoid the algorithm from becoming stuck in local extreme areas, and thus Partial Neighbor Search for optimization finds the optimal solution much faster than the other two algorithms. For both sampling and optimization, we demonstrate the superior performance of the Rejection-Free and Partial Neighbor Search algorithms by applying these methods to several examples, such as the Ising mode, the QUBO question, the Knapsack problem, the 3R3XOR problem, the quadratic programming, etc.

To my parents.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, Professor Jeffrey S. Rosenthal, for his invaluable guidance, mentorship, and support throughout my studies. Professor Rosenthal provided me with a relaxed and stimulating research environment, allowing me to pursue my interests and develop as a researcher. His insightful comments and suggestions have significantly improved the quality of my work.

I would also like to extend my sincere thanks to Professor Ali Sheikholeslami and Professor Michael Evans, both of whom served as members of my thesis committee. Their comments, feedback, and suggestions were instrumental in shaping the direction and content of this thesis.

I would like to express my deepest appreciation to my parents, Haiyan Chen and Sujuan Zhang, for their unwavering love and support throughout my academic journey. Their encouragement and guidance have been invaluable in helping me overcome obstacles and achieve my goals.

I am also grateful to my peers, Miaoshiqi Liu, Siyue Yang, and Kevin Zhang, for their friendship, support, and collaboration throughout my Ph.D. program. Together, we shared ideas, discussed our research, and worked hard to achieve our goals.

I would like to express my heartfelt thanks to my close friends, including Jiayi Liu, Zihan Chen, Xingyu Zhang, Fanzhe Meng, and others, for their emotional support, entertainment, and care. Their presence in my life has made my journey more colorful and enjoyable.

Finally, I would like to acknowledge the financial support provided by Fujitsu Ltd. and Fujitsu Consulting (Canada) Inc., and thank Professor Sheikholeslami for arranging the meetings between us and the Fujitsu team. I am grateful to Hirotaka Tamura and Aki Dote for their engineering insights and great ideas.

Thank you all for your contributions and support in making this thesis possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Monte Carlo method involves the deliberate use of random numbers in a calculation with the structure of a stochastic process [46]. Monte Carlo techniques are based on repeating experiments sufficiently many times to obtain many quantities of interest using the Law of Large Numbers and other statistical inference methods [50]. The three main applications of Monte Carlo methods are sampling, optimization, and numerical integration[50]. This thesis focuses on the Markov chain Monte Carlo method for both sampling and optimization.

From the sampling perspective, the Markov chain Monte Carlo method (MCMC) simulates observations from a target distribution to obtain a chain of states that eventually converges in distribution to the target density itself. The Metropolis-Hastings algorithm [39], [59], an MCMC method, is one of the most popular techniques among its kind [43]. The Metropolis-Hastings algorithm produces a Markov chain $\{X_0, X_1, X_2, \dots\}$ on the state space $\mathcal{S}$ and target density function π, as follows: given the current state x_k, the Metropolis algorithm first proposes a new state y from a proposal distribution $\mathcal{Q}(x_k, \cdot)$; it then accepts the new state by setting $x_{k+1} = y$ with probability $\min\left(1, \frac{\pi(y)\mathcal{Q}(y,x_k)}{\pi(x_k)\mathcal{Q}(x_k,y)}\right)$; otherwise, it rejects the proposal by setting $x_{k+1} = x_k$. This simple algorithm ensures that the Markov chain has π as a stationary distribution. A more detailed review will be given in Section 2.3.

However, the Metropolis-Hastings algorithm may suffer from the inefficiency of rejections. We have a probability of $\left[1 - \min\left(1, \frac{\pi(y)\mathcal{Q}(y,x_k)}{\pi(x_k)\mathcal{Q}(x_k,y)}\right)\right]$ to remain at the current state, even though we have spent time proposing a state, computing a ratio of target probabilities, generating a random variable, and deciding not to accept the proposal. Therefore, we proposed the Rejection-Free algorithm in Chapter 3 to improve the Metropolis algorithm's performance

Furthermore, the parallel hardware can significantly increase the efficiency of Rejection-Free. The use of parallel in Rejection-Free combined with simple techniques such as parallel tempering can yield 100x to 10,000x speedups [76]. It is possible to use processors designed for general purposes, such as Intel and AMD cores, for parallel computing to accelerate the algorithm to some extent. However, these chips were not built for parallel computing, and off-chip communication significantly slows the data transfer rate to and from the cores [77]. On the other hand, parallel hardware designed specifically for MCMC trials has been proposed. For example, the second generation of Fujitsu Digital Annealer uses a dedicated processor called Digital Annealing Unit (DAU) [57] to achieve high speed. This dedicated processor is designed to minimize communication overhead in

arithmetic circuitry and with memory. On the other hand, there is a limit to the number of parallel tasks that can be executed simultaneously on most specialized parallel hardware. For example, the current version of Digital Annealing Unit (DAU) [57] can process the Rejection-Free algorithm efficiently up to 8,192 neighbors simultaneously with on-chip memory, and otherwise, Rejection-Free will be significantly slowed down due to the use of external memory. Accordingly, the ceiling on the number of neighbors that can be evaluated at each step can be a problem. Consequently, we present an enhanced version of the Rejection-Free called Partial Neighbor Search (PNS) in Chapter 4, which only considers part of the neighbors when applying the Rejection-Free technique, whereas the Rejection-Free technique means considering all selected neighbors and calculating the next state when ignoring any immediately repeated states.

In addition, MCMC for optimization is also important. For optimization questions, our goal is no longer to generate a set of samples to approximate a certain distribution, we turn our attention to finding one optimal solution given a set of constraints or limitations. After Metropolis and Hastings proposed the Metropolis-Hasting algorithm [39], [59], Kirkpatrick et al. [47] introduced an interesting variation called the Simulated Annealing algorithm, which can be used to find the optimal values for optimization questions. Regarding the implementation, the Simulated Annealing algorithm is different from the Metropolis-Hastings algorithm by adding temperatures that keep decreasing along the process. Thus, for the high-temperature cases at the very beginning of the Markov chain, the chain accepts the proposed new state with a higher probability, allowing the chain to move around. As the temperature decreases, the probability of accepting the proposed states with target density values lower than the current state becomes low, and the Markov chain will eventually converge to a local optimum, and hopefully, such local optimum solution is close to the global optimum. Therefore, the Simulated Annealing algorithm may also be inefficient with respect to rejections, especially when the temperature becomes low. In order to improve the performance of Simulated Annealing by reducing rejections, we adopt the Rejection-Free algorithm for sampling into an optimization version in Chapter 5. Additionally, Rejection-Free for optimization may also experience inefficiency when it enters local extreme areas. Therefore, we propose Partial Neighbor Search(PNS) again in the optimization version to further enhance its efficiency in Chapter 5. Unlike the sampling problems, even when applied to a single-core implementation, PNS is much more efficient in many optimization problems compared to Rejection-Free, while Rejection-Free is better than the Simulated Annealing.

Our PNS algorithm was originally designed to assist Fujitsu Limited, a Japanese multinational information and communications technology equipment and services corporation that designed DAU [57], to apply sampling or optimization algorithms on the parallel hardware efficiently. As discussed before, DAU [57] can only handle a limited number of tasks, and thus it cannot perform the full Rejection-Free algorithm in parallel on large state spaces. Therefore, we proposed PNS where the number of neighbors being considered at each step can be controlled. Fujitsu Limited has recently applied for several patents for PNS. Just like Rejection-Free, PNS for both sampling and optimization also has a limitation of 8,192 neighbors being considered at each step for on-chip memory. However, PNS can be applied to cases with infinitely many neighbors, and we only need to pick partial neighbor sets with at most 8,192 neighbors. Thus, the PNS algorithm has great utility in extending DAU technology to even more significant problems in an industrial setting, and our algorithm is already well-used in an industrial setting.

For Chapter 2, we first review the basic definitions and theorems for the Markov chains in Section 2.1. Then we review the meaning of the Monte Carlo method by two examples in Section 2.2. Finally, we review the MCMC and the well-known Metropolis-Hastings algorithm as well as their variant in Section 2.3.

Then for Chapter 3, we try to solve the problem of rejection by first proposing a new algorithm called the Uniform Selection algorithm in Section 3.1. However, such the algorithm will not converge in distribution to the target density, and thus we introduce a more promising algorithm called Rejection-Free. Section 3.2 to Section 3.4 talks about the properties and convergence theorems for Rejection-Free as well as its application in the Metropolis-Hastings algorithm. Then Section 3.5 introduces how to adapt Rejection-Free to hybrid chain cases, and Section 3.6 applied Rejection-Free to parallel tempering. Then in Section 3.7, we used two examples, the Bayesian inference, and the Ising model, to illustrate the performance of Rejection-Free sampling questions. Finally, in Section, 3.8, we introduce an efficient method to help us sample proportionally, especially for parallel hardware. For this chapter and the corresponding paper [69], I only did the simulation part.

Moreover, for Chapter 4, in Section 4.1, we introduce our Basic Partial Neighbor Search (Basic PNS) sampling algorithm, which considers subsets of neighbor states for possible moves and calculates the multiplicity list directly from the subsets. Unfortunately, this version of the Markov chain does not converge in distribution to the target density. In Section 4.2, we introduce our unbiased version of Partial Neighbor Search (Unbiased PNS), where the sampling distribution will converge in distribution to the target density correctly; see Section 4.5 for the proof. Unlike Rejection-Free, Unbiased PNS can always use the advantage of the parallel hardware to improve the sampling efficiency, no matter the dimension of the problem. We apply the Unbiased PNS to the QUBO question to illustrate its performance in Section 4.3. In addition, we discuss the choice of subsets of the Unbiased PNS for the QUBO question in Section 4.6. We further illustrate that we can apply the Unbiased PNS to continuous models, and we compare the Metropolis algorithm and Unbiased PNS in a continuous example called the Donuts example to demonstrate the performance of Unbiased PNS in Section 4.4.

In addition, for chapter 5, we introduce what is optimization questions and review the Simulated Annealing in Section 5.1. Following that, Section 5.2 describes how to use the Rejection-Free algorithm to solve optimization problems. Our next point is that the local maximum may lead to another kind of inefficiency for Rejection-Free, and Section 5.3 introduces our Partial Neighbor Search (PNS) algorithm for optimization, which considers just subsets of neighbor states for possible moves. In Section 5.4, we demonstrate how PNS can be applied to quadratic unconstrained binary optimization (QUBO) questions and its effectiveness in solving them. We then discuss why this improvement occurs (Section 5.5), and how its subsets of partial neighbors should be chosen (Section 5.6), as well as its relation to the Tabu Search algorithm (Section 5.7). Moreover, we present several other examples, such as the Knapsack problem (Section 5.8) and the 3R3XOR problem (Section 5.9), to illustrate the advantages of the PNS algorithm in discrete optimization problems. Furthermore, Section 5.10 illustrates another advantage of PNS over Rejection-Free by providing a continuous optimization example known as quadratic programming. PNS can easily be adapted to the general state space by selecting only a finite subset, and it outperforms Simulated Annealing, whereas Rejection-Free cannot be applied in this case due to the need to consider all neighbors at each step. Furthermore, since the performance of PNS in optimization questions is much better

than Rejection-Free and Simulated Annealing, we can adapt the Optimization PNS and use it as the burn-in part for sampling in Chapter 5.11. Geyer [33] stated that burn-in until converging to stationarity is not necessary for MCMC. If we take Geyer's [33] argument, then we can use Optimization PNS to replace the burn-in. On the other hand, we can combine the Optimization PNS and the regular burn-in to get a better algorithm that will converge in distribution to the target density faster.

Finally, in Chapter 6, we summarize our whole thesis, talk about the potential future works and illustrate why our thesis is useful.

Chapter 2

Background Review

2.1 Markov chain Review

Markov chain, made by Andrei Andreevich Markov, is a milestone in the probability theory. In the Bernoulli model, the outcome from the previous events does not change the outcome of current or future events, and a colleague of Markov, namely Pavel Nekrasov started to assume that independence is a condition for the law of large numbers [28]. After the dispute with Pavel, Markov started to address the issue of dependent variables and the law of large numbers[28], [40]. As the first paper for the Markov chain, Markov showed that the average from the states of the Markov chain will converge correctly to the target fixed values under certain conditions [55]. Later, the Markov chain has been applied to a wide variety of problems such as the study of DNA sequences [82], the analysis of economic systems [17], and the design of algorithms in machine learning and artificial intelligence [3].

In this section, we review the basic definitions and the most important theorems such as the Markov chain convergence theorem as well as their proofs for the Markov chain. Note that, the theorems and proofs for this section follow the book *A First Look at Stochastic Processes* [68].

2.1.1 Markov Chain Definitions

Definition 2.1.1. *A (discrete-time, discrete-space, time-homogeneous) Markov Chain is specified by three ingredients:*

1. *a state space $\mathcal{S}$, any non-empty finite or countable set;*
2. *initial probabilities $\{v_i\}_{i\in\mathcal{S}}$, where v_i represents the probability of starting at i (at time 0), so $v_i \geq 0, \forall i \in \mathcal{S}$, and $\sum_{i\in\mathcal{S}} v_i = 1$;*
3. *transition probabilities $\{p_{i,j}\}_{i,j\in\mathcal{S}}$, where $p_{i,j}$ is the probability of jumping to j if you start from i, so $p_{i,j} \geq 0$, $\forall i, j \in \mathcal{S}$, and $\sum_{j\in\mathcal{S}} p_{i,j} = 1$; we use the notation $p_{i,j}^{(n)}$ to represent the transition probability from state i to state j in exactly n jumps.*

Example 1. ***Travel Merchant.*** *Suppose there is a traveling merchant, and at the beginning of each month, the merchant is either residing in the current city or moving to adjacent cities. The map of the cities and probabilities of moving are shown in figure 2.1. Accordingly, a Markov*

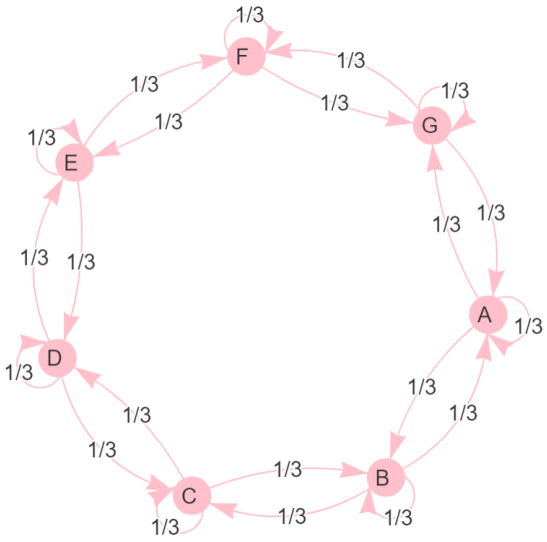

Figure 2.1: The Traveling Merchant, a starting example for the Markov chain

chain is formed by the merchant's stays in each city for each month. The state space contains 7 states $\mathcal{S} = \{A, B, C, D, E, F, G\}$. *Each month, the merchant either stays in the same city or moves to adjacent cities, each with a probability of* $1/3$. *This leads to the transition probabilities as* $p_{A,B} = 1/3$, $p_{B,A} = 1/3$, $p_{A,A} = 1/3$, $\ldots$

Let X_n represent the state of the Markov chain at time n. As a result, $X_0, X_1, X_2, \ldots$ are random variables. The initial state of the Markov chain is X_0, so $\mathbf{P}(X_0 = i) = v_i$, $\forall i \in \mathcal{S}$. Additionally, the transition probabilities can be interpreted as conditional probabilities $p_{i,j} = \mathbf{P}(X_{n+1} = j \mid X_n = i)$, $\forall i, j \in \mathcal{S}$, $n = 0, 1, 2, \ldots$. In other words, the probability at time $n+1$ depends only on the state at time n, called the Markov property.

Furthermore, we can calculate the joint probabilities based on conditional probabilities. For instance, $\mathbf{P}(X_0 = i, X_1 = j) = \mathbf{P}(X_0 = i)\mathbf{P}(X_1 = j \mid X_0 = i) = v_i p_{i,j}$. Similarly, $\mathbf{P}(X_0 = i_0, X_1 = i_1, X_2 = i_2, \ldots X_n = i_n) = v_{i_0} p_{i_0,i_1} p_{i_1,i_2} \ldots p_{i_{n-1},i_n}$. This completely defines the probabilities of the sequence $\{X_n\}_{n=0}^{\infty}$

Now let $\mu_i^{(n)} = \mathbf{P}(X_n = i)$ be the probabilities at time n. Then $\mu_i^{(0)} = v_i$. Moreover, let $s = |\mathcal{S}|$ be the number of elements in $\mathcal{S}$, which can be infinite. We can write

$$\boldsymbol{\mu}^{(n)} = \left(\boldsymbol{\mu}_A^{(n)}, \boldsymbol{\mu}_B^{(n)}, \boldsymbol{\mu}_C^{(n)}, \ldots\right) \tag{2.1}$$

as a $1 \times s$ row vector. In addition, we can also define

$$\boldsymbol{p} = \begin{pmatrix} p_{A,A} & p_{A,B} & p_{A,C} & \cdots \\ p_{B,A} & p_{B,B} & p_{B,C} & \cdots \\ p_{C,A} & p_{C,B} & p_{C,C} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{2.2}$$

as an $s \times s$ matrix. Then we have $\boldsymbol{\mu}^{(n)} = \boldsymbol{\mu}^{(0)}\boldsymbol{p}^n$ in terms of matrix multiplication.

Example 1 (continuing from p. 5)**.** *For the Travel Merchant example as shown in Figure 2.1, we have*

$$\boldsymbol{p} = \begin{pmatrix} 1/3 & 1/3 & 0 & 0 & 0 & 0 & 1/3 \\ 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 1/3 & 0 & 0 & 0 & 0 & 1/3 & 1/3 \end{pmatrix}. \tag{2.3}$$

If we let the merchant start his journey at state A, then

$$\boldsymbol{\mu}^{(0)} = \big(1, 0, 0, 0, 0, 0, 0\big). \tag{2.4}$$

As a result, we can easily calculate the probabilities distribution of the merchant's destination at month n by $\boldsymbol{\mu_i}^{(n)} = \boldsymbol{\mu}^{(0)}\boldsymbol{p}^n$.

Proposition 2.1.1. ***Chapman-Kolmogorov Inequality.*** *$p_{i,j}^{(m+n)} \geq p_{i,k}^{(m)} p_{k,j}^{(n)}$ for any fixed $k \in \mathcal{S}$, $p_{i,j}^{(m+n+s)} \geq p_{i,k}^{(m)} p_{k,\ell}^{(n)} p_{\ell,j}^{(s)}$ for any fixed $k, l \in \mathcal{S}$, etc.*

Proof. Immediate by $\boldsymbol{p}^{(m+n)} = \boldsymbol{p}^{(m)}\boldsymbol{p}^{(n)}$. □

2.1.2 Recurrence and Transience

Before we define the Markov chain to be either recurrent or transient, we first define several shorthand notations:

1. $p_{i,j}^{(n)} = \mathbf{P}(X_n = j \mid X_0 = i)$, which means the probability that the Markov chain moves from state i to state j in n steps; thus $p_{i,j}^{(1)} = p_{i,j}$, where $p_{i,j}$ represents the element at the i-th row and j-th column from matrix $\boldsymbol{p}$;
2. $\mathbf{P}_i(\dots) = \mathbf{P}(\cdots \mid X_0 = i)$; for example, $\mathbf{P}_i(X_n = j) = p_{i,j}^{(n)}$;
3. $\mathbb{E}_i(\dots) = \mathbb{E}(\cdots \mid X_0 = i)$, where $\mathbb{E}$ represents the expectation;
4. $N(i) = \#\{n \geq 1 : X_n = i\}$ represents the total number of times that the chain hits i (without counting time 0);
5. $f_{i,j} := \mathbf{P}_i(\exists n \geq 0 \text{ s.t. } X_n = j) = \mathbf{P}_i(N(j) \geq 1)$, which is the return probability from i to j.

For the above notations, we have the following trivial properties immediately from the definitions:

1. $1 - f_{i,j} = P_i(X_n \neq j, \forall n \geq 1)$, which represents the probability that the Markov chain never hits state j after state i;
2. $\mathbf{P}_i(\text{the chain will visit } j, \text{ and then eventually visit k}) = f_{i,j} f_{j,k}$;
3. $\mathbf{P}_i(N(i) \geq k) = f_{i,i}^k$, and $\mathbf{P}_i(N(j) \geq k) = f_{i,j} f_{j,j}^{k-1}$;
4. $f_{i,k} \geq f_{i,j} f_{j,k}$

Definition 2.1.2. *A state i of a Markov chain is recurrent if $f_{i,i} = 1$, i.e. $\mathbf{P}_i(\exists n \geq 1 \text{ s.t. } X_n = i)$. Otherwise, if $f_{i,i} < 1$, then i is transient.*

Definition 2.1.3. *A state i of Markov chain communicates with state j written $i \to j$, if $f_{i,j} > 0$.*

Definition 2.1.4. *A Markov chain is irreducible if $f_{i,j} > 0$, $\forall i, j \in \mathcal{S}$.*

Example 2. ***Travel Merchant on Integer Axis.*** *The traveling merchant on the integer axis with notation $\{A_i\}_{i\in\mathbb{Z}}$ presenting the cities. Suppose the merchant starts at city A_0, and at each month n, the merchant moves to the cities with the subscript number either $+1$ or -1 with probabilities p and $1-p$. Then the Markov chain is recurrent if and only if $p = 0.5$.*

To make this review section not so redundant, we show the final theorems for the recurrence and transient directly here. For the proof of these two theorems and all related lemmas and propositions, please see [68].

Theorem 2.1.2. *If a chain is irreducible, the following are equivalent:*

1. *There are $k, \ell \in \mathcal{S}$ with $\sum_{n=1}^{\infty} p_{k,\ell}^{(n)} = \infty$*
2. *For all $i, j \in \mathcal{S}$, we have $\sum_{n=1}^{\infty} p_{i,j}^{(n)} = \infty$.*
3. *There is $k \in \mathcal{S}$ with $f_{k,k} = 1$, i.e. with k recurrent.*
4. *For all $j \in \mathcal{S}$, we have $f_{j,j} = 1$, i.e. all states are recurrent.*
5. *For all $i, j \in \mathcal{S}$, we have $f_{i,j} = 1$.*
6. *There are $k, \ell \in \mathcal{S}$ with $\mathbf{P}_k(N(\ell) = \infty) = 1$.*
7. *For all $i, j \in \mathcal{S}$, we have $\mathbf{P}_i(N(j) = \infty) = 1$.*

Theorem 2.1.3. *If a chain is irreducible, the following are equivalent:*

1. *For all $k, \ell \in \mathcal{S}$, $\sum_{n=1}^{\infty} p_{k,\ell}^{(n)} < \infty$.*
2. *There is $i, j \in \mathcal{S}$ with $\sum_{n=1}^{\infty} p_{i,j}^{(n)} < \infty$.*
3. *For all $k \in \mathcal{S}$, $f_{k,k} < 1$, i.e. k is transient.*
4. *There is $j \in \mathcal{S}$ with $f_{j,j} < 1$, i.e. some state is transient.*
5. *There are $i, j \in \mathcal{S}$ with $f_{i,j} < 1$.*
6. *For all $k, \ell \in \mathcal{S}$, $\mathbf{P}_k(N(\ell) = \infty) = 0$.*
7. *There are $i, j \in \mathcal{S}$ with $\mathbf{P}_i(N(j) = \infty) = 0$.*

2.1.3 Markov Chain Convergence

Now we have a rudimentary understanding of the Markov chain, and let us shift our attention to the long-run probabilities $\lim_{n\to\infty} \mathbf{P}(X_n = j)$. Here we only prove the Markov chain convergence theorem for the discrete case. The proof of the convergence theorem for the general state space version is too long to be reviewed here, so please see [61] for details. Later, we will also prove the convergence theorem for our algorithm in Section 4.5.

Definition 2.1.5. *For a probability distribution $\boldsymbol{\pi}$ on $\mathcal{S}$, $\boldsymbol{\pi}$ is called stationary for a Markov chain with transition probabilities $\{p_{i,j}\}$ if $\sum_{i\in\mathcal{S}} \pi_i p_{i,j} = \pi_j$, $\forall j \in \mathcal{S}$. We can also write this formula in matrix notation as $\boldsymbol{\pi}\boldsymbol{p} = \boldsymbol{\pi}$. Equivalently: $\boldsymbol{\pi}$ is a left eigenvector for the matrix $\boldsymbol{p}$ with eigenvalue 1.*

Intuitively, if the chain starts with probabilities $\{\pi_i\}$, then it will always keep the same probabilities. In particular, if the starting distribution $v = \boldsymbol{\pi}$, then $\mu^{(n)} = \boldsymbol{\pi}\boldsymbol{p}^n = \boldsymbol{\pi}$, $\forall n \in \mathbb{N}$.

Example 1 (continuing from p. 5)**.** *For the Travel Merchant example as shown in Figure 2.1, $\boldsymbol{\pi}$ is the uniform distribution on $\mathcal{S}$, i.e. $\pi_i = \frac{1}{6}$ for all $i \in \mathcal{S}$. We can check that $\pi_i \geq 0$ and $\sum_i \pi_i = 1$. Suppose, for instance, for state A, then $\sum_{i\in\mathcal{S}} \pi_i p_{i,A} = \pi_F p_{F,A} + \pi_A p_{A,A} + \pi_B p_{B,A} = \frac{1}{6}(\frac{1}{3}) + \frac{1}{6}(\frac{1}{3}) + \frac{1}{6}(\frac{1}{3}) = \frac{1}{6} = \pi_A$. Similarly, we can prove that $\forall j \in \mathcal{S}$, $\sum_{i\in\mathcal{S}} \pi_i p_{i,j} = \pi_j$. So, $\boldsymbol{\pi}$ is a stationary distribution.*

To find the stationary distributions, or decide if they exist, one helpful technique is as follows:

Definition 2.1.6. *A Markov chain is reversible with respect to a probability distribution $\{\pi_i\}$ if $\pi_i p_{i,j} = \pi_j p_{j,i}$, $\forall i, j \in \mathcal{S}$.*

The usefulness of reversibility is:

Proposition 2.1.4. *If a chain is reversible with respect to $\boldsymbol{\pi}$, then $\boldsymbol{\pi}$ is a stationary distribution.*

If the state space $\mathcal{S}$ is infinite, can there be a stationary distribution? Yes!

Example 3. *Let $\mathcal{S} = \mathbb{N} = \{1, 2, 3, \dots\}$ with $\pi_i = 2 \times 3^{-i}$ for $\forall i \in \mathcal{S}$. We can check that $\pi_i \geq 0$ and $\sum_{i\in\mathcal{S}} \pi_i = 1$.. In addition, the transition probabilities are specified by:*

- $p_{1,1} = p_{2,2} = \frac{9}{10}$;
- $p_{i,i+2} = \frac{1}{10}$, $\forall i \in \mathbb{N}$;
- $p_{i,i-2} = \frac{9}{10}$, $\forall i > 2$;
- $p_{i,j} = 0$, *for all other* $i, j \in \mathbb{N}$.

Then we have $\forall i \in \mathcal{S}$, $\pi_i p_{i,i+2} = 2 \times 3^{-i}(1/10) = 2 \times 3^{-i-2}(9/10) = \pi_{i+2} p_{i+2,i}$. Moreover, $\pi_i p_{i,j} = 0$ if $i \neq j$ and $|j - i| \neq 2$. Therefore, the chain is reversible with respect to $\boldsymbol{\pi}$, and thus $\boldsymbol{\pi}$ is a stationary distribution.

However, does Example 3 have the property that $\lim_{n\to\infty} \mathbf{P}(X_n = j) = \pi_j = 2 \times 3^{-j}$, $\forall j \in S$? The answer is no! For example, $\boldsymbol{\pi}'$, where $\pi'_i = 8 \times 9^{-i}$ for odd i and $\pi'_i = 0$ for even i, is also a stationary distribution. This illustrates that $\lim_{n\to\infty} \mathbf{P}(X_n = j)$ may converge to other distributions even if $\boldsymbol{\pi}$ is a stationary distribution.

Proposition 2.1.5. *If a Markov chain's transition probabilities have* $\lim_{n\to\infty} p_{i,j}^{(n)} = 0$ *for all* $i, j \in S$*, then the chain does NOT have a stationary distribution.*

Proof. If there were such stationary distribution $\boldsymbol{\pi}$, then we would have $\pi_j = \sum_{i\in S} \pi_i p_{i,j}^{(n)}$, $\forall n \in \mathbb{N}$. Thus,

$$\pi_j = \lim_{n\to\infty} \pi_j = \lim_{n\to\infty} \sum_{i\in\mathcal{S}} \pi_i p_{i,j}^{(n)}. \tag{2.5}$$

In addition, since $\sum_{i=1}^{\infty} \sup_n |\pi_i p_{i,j}^{(n)}| \le \sum_{i=1}^{\infty} |\pi_i| = 1 < \infty$, thus we can exchange the sum and limit. Hence,

$$\pi_j = \lim_{n\to\infty} \sum_{i\in\mathcal{S}} \pi_i p_{i,j}^{(n)} = \sum_{i\in\mathcal{S}} \lim_{n\to\infty} \pi_i p_{i,j}^{(n)} = \sum_{i\in\mathcal{S}} (0) = 0. \tag{2.6}$$

Thus, $\pi_j = 0$, $\forall j \in \mathcal{S}$. However, this means that $\sum_j \pi_j = 0$, which is a contradiction. Therefore, there is no stationary distribution. □

Lemma 2.1.6. *If* $\exists k, \ell \in \mathcal{S}$ *with* $\lim_{n\to\infty} p_{k,\ell}^{(n)} = 0$*, then* $\forall i, j \in \mathcal{S}$ *with* $k \to i$ *and* $j \to \ell$*,* $\lim_{n\to\infty} p_{i,j}^{(n)} = 0$*.*

Proof. Since $k \to i$ and $j \to \ell$, we can find $n_1, n_2 \in \mathbb{N}$ with $p_{k,i}^{(n_1)} > 0$ and $p_{j,\ell}^{(n_2)} > 0$.

Since $p_{k,\ell}^{(n_0+n_1+n_2)} \ge p_{k,i}^{(n_1)} p_{i,j}^{(n_0)} p_{j,\ell}^{(n_2)}$, thus for $n = n_0 + n_1 + n_2$, $\lim_{n\to\infty} p_{k,\ell}^{(n)} = 0$ must imply $\lim_{n_0\to\infty} p_{i,j}^{(n_0)} = 0$ □

Corollary 2.1.7. *An irreducible and transient Markov chain cannot have a stationary distribution.*

Proof. If a chain is irreducible and transient, then by the Transience Equivalences Theorem (2.1.3), $\sum_{n=1}^{\infty} p_{i,j}^{(n)} < \infty$ for all $i, j \in S$. Thus we must have $\lim_{n\to\infty} p_{i,j}^{(n)} = 0$ for all $i, j \in S$. Thus, by Proposition 2.1.5, there is no stationary distribution. □

Theorem 2.1.8. *If a Markov chain is irreducible and has a stationary distribution, then it is recurrent.*

Proof. Corollary 2.1.7 says that a chain cannot be irreducible and transient and have a stationary distribution. □

Definition 2.1.7. *The period of a state* i *is the greatest common divisor of the set* $\{n \ge 1, p_{i,i}^{(n)} > 0\}$*. If the period of every state is 1, then we call this Markov chain aperiodic.*

Proposition 2.1.9. *If a state* i *has* $f_{ii} > 0$ *and is aperiodic, then there is* $N_i \in \mathbb{N}$ *such that* $p_{i,i}^{(n)} > 0$ *for all* $n \ge N_i$*.*

Proof. Let $\mathcal{A} = \left\{ n \ge 1 : p_{ii}^{(n)} > 0 \right\}$. Note that $\mathcal{A}$ is non-empty since $f_{i,i} > 0$.

First of all, if $m \in \mathcal{A}$ and $n \in \mathcal{A}$, then $p_{i,i}^{(m)} > 0$ and $p_{i,i}^{(n)} > 0$, so $p_{i,i}^{(m+n)} \ge p_{i,i}^{(m)} p_{i,i}^{(n)} > 0$, which implies $m + n \in \mathcal{A}$. This shows that $\mathcal{A}$ satisfies additivity. In addition, by aperiodicity, $\gcd(A) = 1$. Therefore, $\exists N_i \in \mathbb{N}$ such that for all $n \ge N_i$ we have $n \in \mathcal{A}$, which proves $p_{i,i}^{(n)} > 0$. □

Proposition 2.1.10. *If a chain is irreducible and aperiodic, then for any states* i*,* $j \in S$*, there is* $N_{i,j} \in \mathbb{N}$ *such that* $p_{i,j}^{(n)} > 0$ *for all* $n \ge N_{i,j}$*.*

Proof. Find N_i as Proposition 2.1.9, and find $m \in \mathbb{N}$ such that $p_{i,j}^{(m)} > 0$. Then let $N_{i,j} = N_i + m$. Thus, if $n \ge N_{i,j}$, then $n - m \ge N_i$, so $p_{i,j}^{(n)} \ge p_{i,i}^{(n-m)} p_{i,j}^{(m)} > 0$. □

Lemma 2.1.11. *If a Markov chain is irreducible and aperiodic and has stationary distribution* $\{\pi_i\}$*, then for all* $i, j, k \in \mathcal{S}$*,*

$$\lim_{n\to\infty} |p_{i,k}^{(n)} - p_{j,k}^{(n)}| = 0. \tag{2.7}$$

Proof. Define a new Markov chain $\{(X_n, Y_n)\}_{n=0}^{\infty}$, with state space $\bar{\mathcal{S}} = \mathcal{S} \times \mathcal{S}$, and transition probabilities $\bar{p}_{(i,j)(k,\ell)} = p_{i,k}p_{j,\ell}$.

Intuitively, the new chain has two independent coordinates, and both of them are copies of the original Markov chain. Thus, this new chain has a joint stationary distribution given by $\bar{\pi}_{i,j} = \pi_i\pi_j$ for $i, j \in \mathcal{S}$. In addition, whenever $n \geq \max[N_{i,k}, N_{j,\ell}]$ from Proposition 2.1.10, we must have $\bar{p}_{(i,j)(k,\ell)}^{(n)} = p_{i,j}^{(n)} p_{k,\ell}^{(n)} > 0$. Hence, this new chain is irreducible. In addition, note that the period for the chain is obviously 1, so the chain is also aperiodic. Thus, by Theorem 2.1.8, the new chain is recurrent.

Next, $\forall i_0 \in \mathcal{S}$ and let $\tau = \inf\{n \geq 0 : X_n = Y_n = i_0\}$ be the first time that the new chain hits the state (i_0, i_0). Since the new chain is irreducible and recurrent, Theorem 2.1.2 says that $\bar{f}_{(i,j)(i_0,i_0)} = 1$. This means that, starting from (i, j), the new chain must eventually hit (i_0, i_0), so $\mathbf{P}_{(i,j)}(\tau < \infty) = 1$.

Then, using the Law of Total Probability, we have

$$\begin{aligned} p_{i,k}^{(n)} &= \mathbf{P}_{(i,j)}(X_n = k) \\ &= \sum_{m=1}^{\infty} \mathbf{P}_{(i,j)}(X_n = k, \tau = m) \\ &= \sum_{m=1}^{n} \mathbf{P}_{(i,j)}(X_n = k, \tau = m) + \mathbf{P}_{(i,j)}(X_n = k, \tau > n) \end{aligned} \tag{2.8}$$

Similarly,

$$p_{j,k}^{(n)} = \sum_{m=1}^{n} \mathbf{P}_{(i,j)}(Y_n = k, \tau = m) + \mathbf{P}_{(i,j)}(Y_n = k, \tau > n) \tag{2.9}$$

On the other hand, if $n \geq m$, then

$$\begin{aligned} \mathbf{P}_{(i,j)}(X_n = k, \tau = m) &= \mathbf{P}_{(i,j)}(\tau = m)\mathbf{P}_{(i,j)}(X_n = k \mid \tau = m) \\ &= \mathbf{P}_{(i,j)}(\tau = m)\mathbf{P}_{(i,j)}(X_n = k \mid X_m = Y_m = i_0) \\ &= \mathbf{P}_{(i,j)}(\tau = m)\mathbf{P}(X_n = k \mid X_m = i_0) \\ &= \mathbf{P}_{(i,j)}(\tau = m)p_{i_0,k}^{(n-m)} \end{aligned} \tag{2.10}$$

Similarly,

$$\mathbf{P}_{(i,j)}\left(Y_n = k, \tau = m\right) = \mathbf{P}_{(i,j)}(\tau = m)p_{i_0,k}^{(n-m)}. \tag{2.11}$$

So, $\mathbf{P}_{(i,j)}(X_n = k, \tau = m) = \mathbf{P}_{(i,j)}(Y_n = k, \tau = m)$.

Hence, for all $i, j, k \in \mathcal{S}$,

$$\begin{aligned}
|p_{i,k}^{(n)} - p_{j,k}^{(n)}| =& |\sum_{m=1}^{n} \mathbf{P}_{(i,j)}(X_n = k, \tau = m) + \mathbf{P}_{(i,j)}(X_n = k, \tau > n) \\
& - \sum_{m=1}^{n} \mathbf{P}_{(i,j)}(Y_n = k, \tau = m) - \mathbf{P}_{(i,j)}(Y_n = k, \tau > n)| \\
=& |\mathbf{P}_{(i,j)}(X_n = k, \tau > n) - \mathbf{P}_{(i,j)}(Y_n = k, \tau > n)| \\
\leq& |\mathbf{P}_{(i,j)}(X_n = k, \tau > n)| + |\mathbf{P}_{(i,j)}(Y_n = k, \tau > n)| \\
\leq& \mathbf{P}_{(i,j)}(\tau > n) + \mathbf{P}_{(i,j)}(\tau > n) = 2\mathbf{P}_{(i,j)}(\tau > n).
\end{aligned} \tag{2.12}$$

Since $\mathbf{P}_{(i,j)}(\tau < \infty) = 1$, we have $\lim_{n\to\infty} 2\mathbf{P}_{(i,j)}(\tau > n) = 2\mathbf{P}_{(i,j)}(\tau = \infty) = 0$. Therefore, $\lim_{n\to\infty} |p_{i,k}^{(n)} - p_{j,k}^{(n)}| = 0$

This Lemma illustrates that after a long time n, the chain forgets the starting point. □

Having the above Propositions and Lemmas, we can now start to prove the Markov chain Convergence Theorem.

Theorem 2.1.12. ***Markov chain Convergence Theorem.*** *If a Markov chain is irreducible and aperiodic and has a stationary distribution $\boldsymbol{\pi}$, then $\lim_{n\to\infty} p_{i,j}^{(n)} = \pi_j$ for all $i, j \in \mathcal{S}$, and $\lim_{n\to\infty} \mathbf{P}(X_n = j) = \pi_j$ for any initial distribution $\boldsymbol{v}$.*

Proof. Let's assume we start with $\boldsymbol{v} = \boldsymbol{\pi}$ first.

$$\begin{aligned}
\lim_{n\to\infty} |p_{i,j}^{(n)} - \pi_j| &= \lim_{n\to\infty} |\sum_{k\in\mathcal{S}} \pi_k \times p_{i,j}^{(n)} - \sum_{k\in\mathcal{S}} \pi_k p_{kj}^{(n)})| \\
&= \lim_{n\to\infty} |\sum_{k\in\mathcal{S}} \pi_k \times (p_{i,j}^{(n)} - p_{k,j}^{(n)})| \\
&\leq \lim_{n\to\infty} \sum_{k\in\mathcal{S}} \pi_k |p_{i,j}^{(n)} - p_{k,j}^{(n)}|.
\end{aligned} \tag{2.13}$$

Note that $\sup_n |p_{i,j}^{(n)} - p_{kj}^{(n)}| \leq 1$, so $\sum_{k\in\mathcal{S}} \sup_n \pi_k |p_{i,j}^{(n)} - p_{k,j}^{(n)}| \leq 2 < \infty$. Thus,

$$\begin{aligned}
\lim_{n\to\infty} |p_{i,j}^{(n)} - \pi_j| &\leq \lim_{n\to\infty} \sum_{k\in\mathcal{S}} \pi_k |p_{i,j}^{(n)} - p_{k,j}^{(n)}| \\
&= \sum_{k\in\mathcal{S}} \pi_k \times \lim_{n\to\infty} |p_{i,j}^{(n)} - p_{k,j}^{(n)}| \\
&= \sum_{k\in\mathcal{S}} \pi_k \times 0, \text{ by Lemma 2.1.11} \\
&= 0.
\end{aligned} \tag{2.14}$$

Hence, $\lim_{n\to\infty} p_{i,j}^{(n)} = \pi_j$ for all $i, j \in S$, as claimed.

Finally, $\forall \boldsymbol{v}$, we have:

$$\begin{aligned}
\lim_{n\to\infty} \mathbf{P}(X_n = j) &= \lim_{n\to\infty} \sum_{i\in\mathcal{S}} \mathbf{P}t(X_0 = i, X_n = j) \\
&= \lim_{n\to\infty} \sum_{i\in\mathcal{S}} v_i p_{i,j}^{(n)} \\
&= \sum_{i\in\mathcal{S}} v_i \lim_{n\to\infty} p_{i,j}^{(n)} \\
&= \sum_{i\in\mathcal{S}} v_i \pi_j \\
&= 1 \times \pi_j \\
&= \pi_j.
\end{aligned} \tag{2.15}$$

□

Corollary 2.1.13. *If a Markov chain is irreducible and aperiodic, then it has at most one stationary distribution.*

2.2 Monte Carlo method Review

The Monte Carlo method is a statistical analysis based on the generation of random objects or processes repeatedly to obtain numerical results [50]. Without using the name of the Monte Carlo method, scientists have been using methods of stochastic sampling back a long time ago. Buffon's needle experiment posed in 1777 is the signature one. Later, Laplace suggested that we can use this experiment to estimate the approximate value of π by throwing needles of length L in a random fashion onto a smooth table ruled with parallel lines separated by a distance of $2L$, and the percentage of needles interacting with one of the ruled lines can be an estimate for $\frac{1}{\pi}$ [74]. After that, the idea of using stochastic sampling methods first occurred to Stanislaw Ulam, who wanted to estimate the success probability of solitaire by playing a large number of games and tallying the number of successful plays [58]. Ulam immediately thought of problems of neutron diffusion and other questions of mathematical physics [23], and then he communicated this idea with John von Neumann, who was working on the theoretical calculations related to the development of thermonuclear weapons [75]. Von Neumann was intrigued by this idea, and they start to think of using random sampling to simulate the behavior of neutron transport to make predictions about the behavior of the atomic bomb by ENIAC [75], the world's first electronic digital computer [81]. Then in 1949, Metropolis and Ulam published their breakthrough paper named "The Monte Carlo method" [60], which was the first unclassified paper on the Monte Carlo methods, and the first to have the name Monte Carlo associated with stochastic sampling [75]. It is named after the Monte Carlo Casino, located in Monaco, where Ulam's uncle borrowed money from relatives for gambling [58].

examples In the years that followed, the Monte Carlo method was used in a wide variety of applications including industrial engineering and operations research, physical processes and materials science, random graphs and combinatorial structures, economics and finance, and computational statistics [50]. Under different scenarios, the Monte Carlo methods can be applied differently but usually follows the following logic:

1. determines the basic elements of the problem such as what random variables are we considering, what value are we estimating, what is the domain for the corresponding random variables, etc;

for example, we can estimate the probability of a given coin being head when tossing, the random variable is the tossing results, the domain is $\{\text{head}, \text{tail}\}$;

2. generates samples randomly by the random variables; for example, we can toss the given coin a certain number of times and record the results;

3. performs necessary computation and aggregates the results; for example, we can calculate the percentage of the heads, and consider it as an estimation of the probability of tossing heads with the given coin.

Here we use some additional examples to illustrate how the Monte Carlo method works further.

Example 4. *Suppose we want to estimate* $\mathbb{E}[\frac{1}{Z^2}\sin(Z)\exp(Z^2)]$*, where* $Z \sim$ *Normal*$(0,1)$*. The classical way of the Monte Carlo solution is to generate* $\{z_1, z_2, \ldots, z_n\}$ *from Normal*$(0,1)$*, and then let* $\frac{1}{n}\sum_{i=1}^{n}\frac{1}{z_i^2}\sin(z_i)\exp(z_i^2)$ *to be an estimate of* $\mathbb{E}[\frac{1}{Z^2}\sin(Z)\exp(Z^2)]$

The Monte Carlo method can also be used to calculate integration. The key to doing so is to re-write the integration as an expectation.

Example 4 (continuing from p. 14)**.** *It can be hard for us to calculate the numerical integration* $\int_{-\infty}^{\infty}\frac{1}{z^2}\sin(z)dz$ *directly. On the other hand, we have*

$$\begin{aligned}\mathbb{E}[\frac{1}{Z^2}\sin(Z)\exp(Z^2)] &= \int_{-\infty}^{\infty}\frac{1}{z^2}\sin(z)\exp(z^2)f(z)dz, \text{ where } f \text{ is the density for } Normal(0,1)\\ &= \int_{-\infty}^{\infty}\frac{1}{z^2}\sin(z)\exp(z^2)\frac{1}{\sqrt{2\pi}}\exp(-z^2)dz\\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\frac{1}{z^2}\sin(z)dz\end{aligned} \tag{2.16}$$

Thus, we can use $\sqrt{2\pi}$ *times the average value* $\frac{1}{n}\sum_{i=1}^{n}\frac{1}{z_i^2}\sin(z_i)\exp(z_i^2)$ *from the previous Markov chain to be an approximation for such integration.*

Example 5. *Suppose we want to calculate* $\int_{x=0}^{1}\int_{y=0}^{1}g(x,y)dydx$*. If we define* $X, Y \sim$ *Uniform*$[0,1]$*, then* $\int_{x=0}^{1}\int_{y=0}^{1}g(x,y)dydx = \mathbb{E}\big(g(X,Y)\big)$*. Thus, if we generate* $\{x_1, x_2, \ldots, x_n\}$ *and* $\{y_1, y_2, \ldots, y_n\}$ *from Uniform*$[0,1]$ *independently, then* $\frac{1}{n}\sum_{i=1}^{n}g(x_i, y_i)$ *is an estimate of such integration.*

2.3 Markov chain Monte Carlo Review

Markov chain Monte Carlo (MCMC) is essentially Monte Carlo integration using Markov chains [34]. Monte Carlo integration draws samples from the required distribution by running a cleverly constructed Markov chain for a long time and then forms sample averages to approximate expectations [34]. As introduced when we reviewed the Monte Carlo method, Ulam came up with the original idea of the Monte Carlo method in 1946 [58], this idea was adopted by von Neumann for the application in neuron diffusion[75], and the name Monte Carlo was suggested by Nicholas Metropolis [58]. All these occurrences are closely related to the very first general-purpose digital computer, the ENIAC, and with it, scientists have invented a lot of methods related to random sampling [75]. The first MCMC algorithm, on the other hand, is associated with a second computer, namely MANIAC, built

in Los Alamos under the direction of Metropolis in early 1952 [12]. Metropolis et al. [59] demonstrated the validity of the first MCMC algorithm named after Metropolis himself. The number of iterations of the Metropolis algorithm used in this paper seems to be limited with only 16 iterations for burn-in and up to 64 subsequent steps for sampling, which took 4-5 hours on the Los Alamos computer [12]. In addition, the Metropolis algorithm has also been generalized by Hastings [39]. Later, Kirkpatrick et al. [47] introduced an interesting variation called the Simulated Annealing algorithm, which can be used to find the optimal values for optimization questions, and we will talk more about the optimization in Chapter 5.

In the MCMC, the Metropolis algorithm has been the most successful and influential one [6]. It is designed to generate a Markov chain that converges to a given target distribution π on a state space $\mathcal{S}$. The Metropolis-Hastings algorithm is a generalized version of the Metropolis algorithm, including the possibility of an asymmetric proposal distribution $\mathcal{Q}$ [43].

Here, we first introduce the essential elements for the discrete sampling questions by MCMC:

1. a state space $\mathcal{S}$, any non-empty finite or countable set;
2. a real-valued target distribution $\pi : \mathcal{S} \to [0, 1]$ where $\sum_{x\in\mathcal{S}} \pi(x) = 1$;
3. $\forall x \in \mathcal{S}$, $\exists$ a proposal distribution $\mathcal{Q}(x, \cdot)$ where $\sum_{y\in\mathcal{S}\setminus\{x\}} \mathcal{Q}(x, y) = 1$, and $\mathcal{Q}(x, y) > 0 \iff \mathcal{Q}(y, x) > 0$, $\forall x, y \in \mathcal{S}$; if $\forall x, y \in \mathcal{S}$, $\mathcal{Q}(x, y) = \mathcal{Q}(y, x)$, then $\mathcal{Q}$ is symmetric;
4. $\forall x \in \mathcal{S}$, $\exists$ a neighbor set $\mathcal{N}(x) \coloneqq \{y \in \mathcal{S} \mid \mathcal{Q}(x, y) > 0\} \subset \mathcal{S}\setminus\{x\}$;
5. initial probabilities $\{v_x\}_{x\in\mathcal{S}}$ represent the probability at time 0, so $v_x \geq 0, \forall x \in \mathcal{S}$, and $\sum_{x\in\mathcal{S}} v_x = 1$; we usually start our MCMC at a certain state by a given state x_0, and then we can set $v_{x_0} = 1$ and $v_x = 0$, $\forall x \in \mathcal{S}\setminus\{x_0\}$.

Here, we focus on discrete cases first, and we will discuss the general state space in Section 4.5. The Metropolis-Hastings algorithm is described in Algorithm 1.

Algorithm 1 the Metropolis-Hastings algorithm

initialize X_0
for k in 1 to K **do**
  random $Y \in \mathcal{N}(X_{k-1})$ based on $\mathcal{Q}(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < \frac{\pi(Y)\mathcal{Q}(Y,X_{k-1})}{\pi(X_{k-1})\mathcal{Q}(X_{k-1},Y)}$ **then**
    ▷ accept with probability $\min\left\{1, \frac{\pi(Y)\mathcal{Q}(Y,X_{k-1})}{\pi(X_{k-1})\mathcal{Q}(X_{k-1},Y)}\right\}$
    $X_k \leftarrow Y$ ▷ accept and move to state Y
  else
    $X_k \leftarrow X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
end for

Algorithm 1 ensures the Markov chain $\{X_0, X_1, X_2, \ldots, X_K\}$ has π as stationary distribution. Assuming irreducibility, it follows that the expected value $E_\pi(h)$ of a functional $h : \mathcal{S} \to \mathbb{R}$ with respect to $\boldsymbol{\pi}$ can be estimated by $\frac{1}{K}\sum_{i=1}^{K} h(X_i)$ for sufficiently large run length K.

2.3.1 Hybrid Chains

Consider k different Markov chain updating schemes $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k\}$ as defined in Equation 2.2, all of which leave π stationary. There are various ways to combine the chains to form a hybrid chain that still leaves π stationary. We can, for example, run the schemes sequentially to produce the systematic-scan Markov chain $\boldsymbol{p} = \boldsymbol{p}_1\boldsymbol{p}_2 \ldots \boldsymbol{p}_k$. As an alternative, we can select one uniformly at random in each iteration, resulting in the random-scan Markov chain $\boldsymbol{p} = (\boldsymbol{p}_1 + \boldsymbol{p}_2 + \cdots + \boldsymbol{p}_k)/k$. Combining strategies such as these can be used to build hybrid Markov chains. Hybrid chains may have superior convergence properties under many circumstances such as [64], [65]. This combining technique also forms the basis of the Gibbs sampler.

For the Metropolis-Hastings algorithm, we can be more specific. An example of the Metropolis-Hastings algorithm with alternating chains for every L_0 step among $\mathcal{I}$ proposal distributions $\mathcal{Q}^0$, $\mathcal{Q}^1$, $\ldots$, $\mathcal{Q}^{\mathcal{I}-1}$ is shown as Algorithm 2. Note that, we used superscripts as the index for the proposal distribution Q^i for different proposals. Later for Partial Neighbor Search, we will be using proposal distributions with subscripts like Q_i, and we use different notations for proposal distributions under different circumstances to illustrate their difference as we will discuss later in Section 4.2.1.

Algorithm 2 Metropolis-Hasting algorithm with Alternating Chains

initialize $i \leftarrow 0$ ▷ start with proposal distribution $\mathcal{Q}^0$
initialize $L \leftarrow L_0$ ▷ start with L_0 remaining samples
initialize X_0
for k in 1 to K **do**
  random Y based on $\mathcal{Q}^i(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < \frac{\pi(Y)\mathcal{Q}^i(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}^i(X_{k-1}, Y)}$ **then**
    ▷ accept with probability $\min\left\{1, \frac{\pi(Y)\mathcal{Q}^i(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}^i(X_{k-1}, Y)}\right\}$
    $X_k \leftarrow Y_k$ ▷ accept and move to state Y
  else
    $X_k \leftarrow X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
  $L \leftarrow L - 1$ ▷ one less remaining sample from the proposal distribution
  if $L = 0$ **then** ▷ if we don't have enough remaining samples
    $i \leftarrow i + 1 \mod \mathcal{I}$ ▷ switch to the next proposal distribution
    $L \leftarrow L_0$ ▷ L_0 remaining states for the next proposal distribution
  end if
end for

2.3.2 Parallel Tempering

Parallel tempering can be traced to a 1986 paper by Swendsen and Wang [79], and they used the name replica Monte Carlo simulation. In this paper, the authors use replicas at different temperatures to study the system of spin-glasses. By partial exchanges of configuration information, the authors greatly reduce the long correlation times characteristic of standard methods, allowing the investigation of lower temperatures with less computer time than previously necessary [79]. Later, Geyer extended this idea and made complete exchanges of configuration information [31]. Initially, applications of the new method were limited to problems in statistical physics [22]. Then Hansmann

[38], Falcioni and Deem [25], and Okamoto and Sugita [78], etc. extended such method to the field of spanning physics, chemistry, biology, engineering, materials science and so on.

The intuition behind parallel tempering is simple. For some sampling questions at low temperatures, the distribution may have very separate but equally important local extreme areas, and it can be hard for us to travel between those local extreme areas. The high-temperature systems are generally able to sample large volumes of phase space, whereas low-temperature systems, which can sample precisely in a local region of phase space, may become trapped in local extreme areas [79]. Thus, we can use two or more Markov chains at different temperatures to make use of the advantages of both high and low temperatures. The Markov chains at different temperatures may swap the current state with each other by some given probability, usually following the Metropolis-Hastings criterion. Thus, the Markov chain at low temperatures is more likely to travel between local maximum areas and samples more efficiently. Although parallel tempering can be applied in a variety of ways, it usually contains the following steps:

1. choose the number of replicas and the corresponding temperatures; usually, temperature $T = 1$ must be chosen since this is our target distribution to sample from; the highest temperature should be chosen to be high enough to explore the entire state space; we may also need several intermediate temperatures since if the difference between $T = 1$ and the highest temperature is too large, the probability of direct swapping between these two replicas is low; the total number of temperatures being chosen should not be too large to make the algorithm computationally infeasible;

2. initialize a Markov chain for each replica, and choose initial states for them;

3. run the Markov chains under different temperatures for a specified number of steps under their respective temperatures;

4. perform exchange between temperatures based on some probability; usually the exchange can only be made between the adjacent temperatures, and the probability of exchange can be based on the Metropolis-Hastings criterion or any other suitable method;

5. repeat step 3 and 4 for sufficient times to make sure the Markov chains are converging to the corresponding distribution;

6. the Markov chain for the temperature $T = 1$ contains the samples which approximate the target distribution; generally, parallel tempering will result in faster convergence than using a single Markov chain with a temperature of 1.

An example algorithm with two temperatures is shown as Algorithm 3. Suppose the two temperatures are T_1 and T_2 where $T_1 = 1$ and the high temperature T_2 is at our choice. Here we define the $\{X_{1,0}, X_{1,1}, X_{1,2}, \dots\}$ and $\{X_{2,0}, X_{2,1}, X_{2,2}, \dots\}$ are the two chains corresponding to the two temperatures accordingly. We use π_1 and π_2 to describe the target density of these two temperatures. $\pi_1 \propto \pi^{1/T_1} = \pi$ is the same as the target density. $\pi_2 \propto \pi^{1/T_2}$ is a flatter target density, and it is easy for the Markov chain to explore the whole state space. Here, we propose exchange after every L steps, and we exchange the states with the Metropolis-Hastings criterion, which swaps the states with probability $p = \min\left[1, \frac{\pi_1(X_2)\pi_2(X_1)}{\pi_1(X_1)\pi_2(X_2)}\right]$.

We will illustrate the power of parallel tempering with a simple example.

Algorithm 3 the Metropolis-Hastings algorithm with parallel tempering of two temperatures

initialize $X_{1,0}$ and $X_{2,0}$, set $T_1 = 1$, and choose T_2 ▷ T_2 is the high temperature at our choice
choose L ▷ we propose exchange after every L states
for k in 1 to K **do**
  random $Y \in \mathcal{N}(X_{k-1})$ based on $\mathcal{Q}(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < \frac{\pi(Y)\mathcal{Q}(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}(X_{k-1}, Y)}$ **then**
    ▷ accept with probability $\min\left\{1, \frac{\pi(Y)\mathcal{Q}(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}(X_{k-1}, Y)}\right\}$
    $X_k \leftarrow Y$ ▷ accept and move to state Y
  else
    $X_k \leftarrow X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
  if $k \mod L = 0$ **then**
    random $U'_k \sim \text{Uniform}(0, 1)$
    if $U'_k < \frac{\pi_1(X_{2,k})\pi_2(X_{1,k})}{\pi_1(X_{1,k})\pi_2(X_{2,k})}$ **then**
      swap $X_{1,k}$ and $X_{2,k}$.
    end if
  end if
end for

the number of steps needed	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
without parallel tempering	7195	151134	452958	3463047	1755254	52555384
with parallel tempering	962	24660	95310	1316368	276162	96479112

Table 2.1: Number of steps needed for Markov chains with and without parallel tempering to get good samples

Example 6. *Suppose* $\mathcal{S} = A, B, C, D, E, F$ *with* $\pi(A) = \pi(F) = 0.4$, $\pi(B) = \pi(E) = 0.099$, $\pi C = \pi(D) = 0.001$. *Suppose the proposal distribution* $\mathcal{Q}$ *is uniform on the adjacent states. If we want to sample from the case with temperature* $T = 1$, *then the probability of the Markov chain moving from state* B *to state* C *with one step is about* $\mathcal{Q}(B, C) \times \frac{\pi(C)}{\pi(B)} = 0.5 \times \frac{0.001}{0.099} \approx 0.005$. *On the other hand, for a high-temperature case such as* $T = 100$, *the corresponding probability is* $\mathcal{Q}(B, C) \times \left[\frac{\pi(C)}{\pi(B)}\right]^{1/100} = 0.5 \times \left[\frac{0.001}{0.099}\right]^{1/100} \approx 0.5$. *We care about the probability from state* B *to state* C *because we have to switch between the two local maximum areas* $\{A, B\}$ *and* $\{E, F\}$ *for many times to make sure the empirical distribution from the samples is close to the target distribution, and thus, we have to move from* B *to* C *for many times.*

In addition, we can also check the converging speed by simulation. Suppose we call samples where the empirical distribution $\mathcal{P}_{sampled}$ *is different from the target distribution for at most* 0.0001 *for all states to be good samples. That is,* $\forall i \in \mathcal{S}$, $|\pi(i) - \mathcal{P}_{sampled}| < 0.0001$. *How many steps do we need to get good samples? We simulate the Markov chain with and without parallel tempering both 100 times and record the number of steps needed to reach good samples. Note that, for the parallel tempering case with two temperatures, two Markov chains with 100 states will be recorded as 200 steps, and swapping between chains will also be considered as one step. The result is shown in Table 2.1.*

Table 2.1 shows that with parallel tempering, the number of steps needed by the Markov chains to reach good samples is significantly reduced.

2.4 Parallel Computing Review

Moore's Law states that the number of transistors on a microchip doubles every two years, though the cost of computers is halved. This law, holding for more than 50 years, is approaching its end, and the improvements from the general processors are slowing down [16]. Thus, the market and the researchers have turned their attention to domain-specific computing that improves performance by applying dedicated hardware and algorithms to specific application domains [57]. Parallel computing is one of this kind. It is the use of multiple calculations or execution threads simultaneously to solve problems [2]. It usually involves dividing large-scale problems into smaller sub-problems that can be solved independently and then distributing these small sub-problems onto different processors. Parallel computing can increase processing speed and improve efficiency for a lot of algorithms, but developing parallel software can be challenging, and requires specialized skills and tools to ensure that the computations are executed efficiently and correctly.

Suppose we want to sample from a target distribution by a programming language like C++ on a personal computer. It is easy for us to write code to generate a single Markov chain to approximate the target distribution. This can be done in serial order by one processor. In addition, if we have several processors, we can use tools like Message Passing Interface (MPI) to help our processors to communicate with each other. For example, for my personal desktop, I can generate 16 Markov chains from the target distribution at the same time, and thus, the sampling efficiency is greatly improved. However, this example is very simple, and the chains being generated by each processor are independent. We almost fully parallelize the program, and thus we don't need frequent communications among processors during the process. But for other problems, some portion of the program cannot be parallelized, and frequent communications among processors is necessary.

According to Amdahl's law, the speedup of a parallel program is limited by the portion of the program that cannot be parallelized. In this case, not only the part of the program that cannot be parallelized but also the communication time used by the processors will take a lot of time. For example, for some parallel computing tasks done by my personal computer, using 16 processors can only reduce 50% of the time needed by the program. For the general-designed processors like Intel and AMD cores, off-chip communication significantly slows down the transfer rate of data to and from the cores [77]. Thus, other people like Fujitsu have designed specialized chips for parallel computing.

For example, the second generation of Fujitsu Digital Annealer uses a dedicated processor called a Digital Annealing Unit (DAU) [57] to achieve high speed. This dedicated processor is designed to minimize communication overhead in arithmetic circuitry and with memory. With fully coupled bit connectivity and high coupling resolution as a major feature, the DAU can be used to solve a wide variety of combinatorial optimization problems [57]. All the algorithms introduced in the thesis were designed purposefully for DAU at the very beginning. On the other hand, we also find these algorithms useful when being applied to serial computing with regular processors. For example, in all later simulations, using serial computing with a single processor, the speed to solve the sampling and optimization question by Rejection-Free and Partial Neighbor Search, as we will introduce later, is faster than the Metropolis-Hastings algorithm and the Simulated Annealing algorithm.

Chapter 3

Jump Markov chains and Rejection-Free Metropolis algorithm

The Metropolis algorithm might reject many proposals during the sampling process, and too many rejections will lead to inefficiencies in convergence. In certain contexts, the optimal Metropolis algorithm should reject over three-quarters of its proposals [30], [66]. After each rejection, the Markov chain will remain at the current state even though we have sampled a proposed state, computed a ratio of target probabilities, and decided not to accept the proposal. These rejections are normally considered to be a necessary evil of the Metropolis algorithm. However, recent technological advances have allowed for exploiting parallel in computer hardware. We are now able to compute all potential acceptance probabilities at once, and thus the Markov chain has the possibility of skipping the rejection steps and instead accepting a move for every step. Such Rejection-Free algorithms can be very efficient, but the algorithm must be executed correctly. Otherwise, the corresponding Markov chain will be biased, as we now explain.

3.1 The Uniform Selection Algorithm

Suppose we are sampling from an MCMC with the current state x and the neighbor set is $\mathcal{N} = \{y_1, y_2, \ldots, y_k\}$ which is a large but finite set, and all y_i's are distinct from x. The proposal distribution is uniformly on the neighbor set. Then, we can do the following for the current step:

1. sample $U \sim \text{Uniform}[0, 1]$;
2. consider the subset of the neighbors $\mathcal{N}_U \coloneqq \{y_i \mid U < \frac{\pi(y_i)}{\pi(x)}\}$ that would have been accepted if given U in the Metropolis algorithm;
3. (a) if $\mathcal{N}_U$ is not empty, pick one of the states in $\mathcal{N}_U$ uniformly at random;
   (b) otherwise, re-start from step 1 and try again.

Technically speaking, the method involves rejection when we move to case 3 (b), though its probability is small. On the other hand, compared to the Metropolis algorithm, this algorithm will mostly

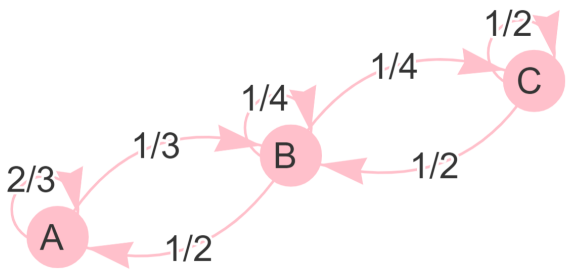

Figure 3.1: The transition probabilities from the Metropolis chain in Example 7

move somewhere else, so there is almost no rejection. However, this algorithm is different from true MCMC, and might not converge to the true density π, as we now show

Example 7. *Suppose the state space $\mathcal{S} = \{A, B, C\}$, with the true density π being $\pi(A) = 1/2$, $\pi(B) = 1/3$, and $\pi(C) = 1/6$. In addition, suppose that from each state x, the chain proposes to move either to the adjacent state with probability $1/2$ each, and if the state has only one adjacent state, then for the other $1/2$, the proposal will always be rejected. We can calculate that the Metropolis algorithm would have Markov chain transition probabilities as in Figure 3.1. The limiting stationary distribution from the Metropolis algorithm is $\pi = (1/2, 1/3, 1/6)$, which is the same as the true density π as they must. However, the Uniform Selection algorithm would have Markov chain transition probabilities as in Figure 3.2, with limiting stationary distribution being $(3/5, 4/15, 2/15)$ which is significantly different from the true density.*

For example, from state B, the Metropolis algorithm would accept a proposed move to state A with probability 1 and would accept a proposed move to state C with probability $\frac{\pi(B)}{\pi(C)} = \frac{1/6}{1/3} = 1/2$, so the Markov chain at state B would be twice as likely to move to state A as to move to state C. However, for the above Uniform Selection version, $\mathcal{N}_U = \{A\}$ for $u > \frac{1}{2}$. Moreover, $\mathcal{N}_U = \{A, C\}$ for $u \leq 1/2$, and the Markov chain would move to state A or state C with probability $1/2$ each. Thus the Markov chain would move to state A with probability $(1/2)(1) + (1/2)(1/2) = 3/4$, and for state C, the probability is $(1/2)(0) + (1/2)(1/2) = 1/4$. That is, from state B, the Markov chain would now be three times as likely to move to state A as to move to state C, not twice. This illustrates that this Uniform Selection algorithm will fail to converge to the true density π but to some other distribution.

Example 8. *Suppose now that the state space is the set $\mathcal{S} = A_0, A_1, A_2, A_3, \ldots$ of all non-negative integers, with target distribution π defined by writing*

$$\pi(4a + b) = \frac{1}{135}\left(\frac{8}{9}\right)^a 2^b, \ b = 0, 1, 2, 3 \text{ and } a = 0, 1, 2, \ldots$$

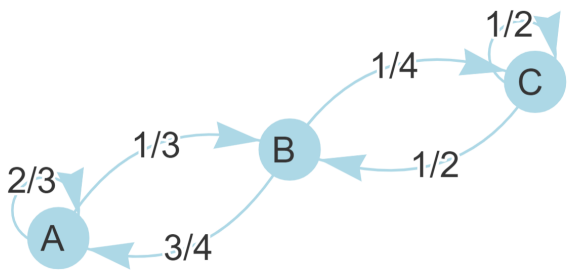

Figure 3.2: The transition probabilities from the Uniform Selection chain in Example 7

As a check,

$$\sum_{A_i \in \mathcal{S}} \pi(A_i) = \sum_{a=0}^{\infty} \frac{1}{135} \left(\frac{8}{9}\right)^a (2^0 + 2^1 + 2^2 + 2^3) = \frac{1}{135} \times \frac{1}{1 - 8/9} \times 15 = 1$$

Thus, π is a valid probability distribution. The Metropolis algorithm chain for this example is given by Figure 3.3, and it has the correct limiting stationary distribution π, as it must. However, the Uniform Selection chain is instead given by Figure 3.4. It is easy to prove that the Uniform Selection chain is transient in this case. For example, if the Uniform Selection chain for this example begins at state A_{4a} for some positive integer $a \geq 2$, then the probability it will ever reach the state A_3 is less or equal to $\left(\frac{8}{9}\right)^{a-1}$, which is strictly smaller than 1. That is, the Uniform Selection chain might fail to ever reach the optimal value. For instance, if $X_0 = A_{100}$, then $a = 25$ and the probability of failure is at least $1 - \left(\frac{8}{9}\right)^{24} > 0.94 = 94\%$.

Examples 7 and Example 8 show that the Uniform Selection algorithm may converge to the wrong limiting distribution, and thus should not be used for sampling purposes.

Furthermore, Example 8 also has implications for optimization. Any Markov chain which gives consistent estimators can be used to find the mode of π, either by running the chain for a long time and taking its empirical sample mode or by keeping track of the largest value $\pi(x)$ over all samples visited. However, Example 8 shows that a Uniform Selection chain could be transient and thus fail to find or converge to the maximum value at all. We will talk more about the optimization questions in Chapter 5.

3.2 The Jump Chain

Due to the problems with the Uniform Selection Algorithm identified above, we instead turn attention to a more promising avenue, the Jump Chain. Our definitions are as follows. Let $\{X_n\}$ be an irreducible Markov chain on a state space $\mathcal{S}$, and we call it the original chain. For ease of exposition, we initially assume that $\mathcal{S}$ is finite or countable, though we later extend this to general Markov chains with densities. To avoid trivialities, we assume throughout that $|\mathcal{S}| > 1$. Given a run $\{X_n\}$ of the Markov chain, we define the Jump Chain $\{J_k\}$ to be the same chain except omitting

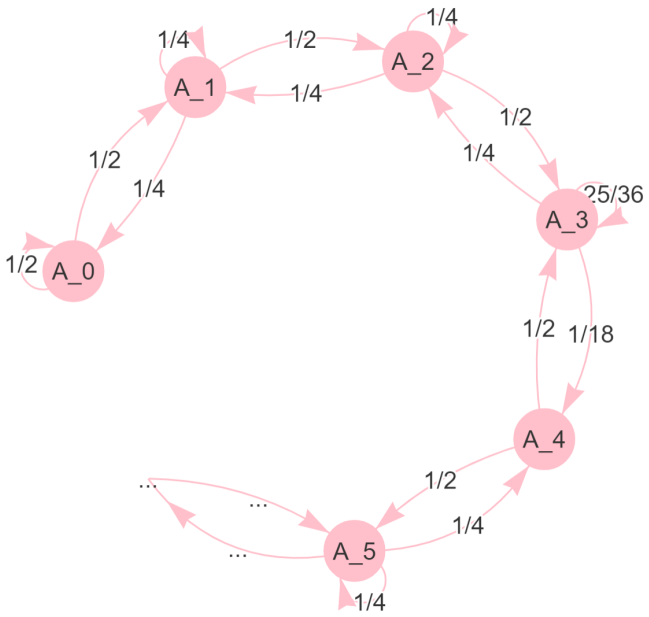

Figure 3.3: The transition probabilities from the Metropolis chain in Example 8

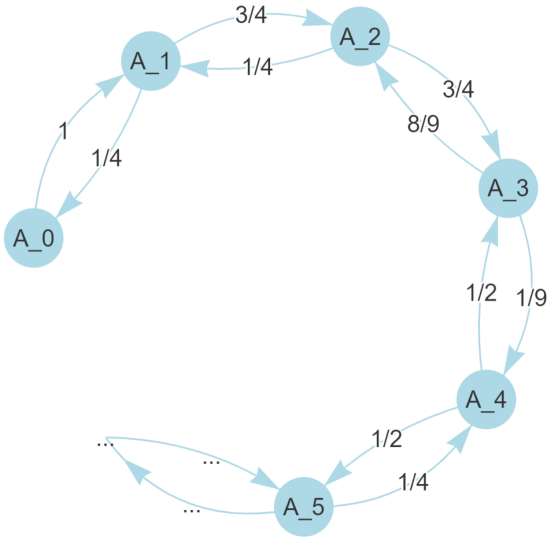

Figure 3.4: The transition probabilities from the Uniform Selection chain in Example 8

any immediately repeated states, and the Multiplicity List $\{M_k\}$ to count the number of times the original chain remains at the same state. For example, if the original chain $\{X_n\}$ was

$$\{X_n\} = \{a, b, b, b, a, a, c, c, c, c, d, d, a, \dots\},$$

then the jump chain $\{J_k\}$ would be

$$\{J_k\} = \{a, b, a, c, d, a, \dots\},$$

and the corresponding multiplicity list $\{M_k\}$ would begin

$$\{M_k\} = \{1, 3, 2, 4, 2, \dots\}.$$

The concept of jump chains arises frequently for Markov processes, especially for continuous time processes where they are often defined in terms of infinitesimal generators; see Section 4.4 of [21] or Proposition 4.4.20 of [68] for examples. Here we develop the essential properties that we will use below. Most of these properties are already known in the context of Metropolis-Hastings algorithms. To continue, define

$$\mathbf{P}(y \mid x) = \mathbf{P}[X_{n+1} = y | X_n = x], \ x, y \in \mathcal{S} \tag{3.1}$$

to be the transition probabilities for the original chain $\{X_n\}$. And, let

$$\alpha(x) = \mathbf{P}[X_{n+1} \neq x | X_n = x] = \sum_{y \neq x} \mathbf{P}(y \mid x) = 1 - \mathbf{P}(x \mid x) \tag{3.2}$$

be the probability that the original chain will move away from x on the next step. Note that since the chain is irreducible and $|\mathcal{S}| > 1$, we must have $\alpha(x) > 0$ for all $x \in \mathcal{S}$. We then verify the following properties of the jump chain.

Proposition 3.2.1. *The jump chain $\{J_k\}$ is itself a Markov chain, with transition probabilities $\hat{\mathbf{P}}(y \mid x)$ specified by $\hat{\mathbf{P}}(x \mid x) = 0$, and for $y \neq x$,*

$$\hat{\mathbf{P}}(y \mid x) := \mathbf{P}[J_{k+1} = y | J_k = x] = \frac{\mathbf{P}(y \mid x)}{\sum_{z \neq x} \mathbf{P}(z \mid x)} = \frac{\mathbf{P}(y \mid x)}{\alpha(x)} \tag{3.3}$$

Proof. It follows from the definition of $\{J_k\}$ that $\hat{\mathbf{P}}(x \mid x) = 0$. For $x, y \in S$ with $y \neq x$, we compute that

$$\begin{aligned} \hat{\mathbf{P}}(y \mid x) &:= \mathbf{P}[J_{k+1} = y | J_k = x] \\ &:= \mathbf{P}[X_{n+1} = y | X_n = x, X_{n+1} \neq X_n] \\ &:= \frac{\mathbf{P}[X_{n+1} = y, X_{n+1} \neq X_n | X_n = x]}{\mathbf{P}[X_{n+1} \neq X_n | X_n = x]} \\ &= \frac{\mathbf{P}(y \mid x)}{\sum_{z \neq x} P(z \mid x)} \\ &= \frac{\mathbf{P}(y \mid x)}{\alpha(x)} \end{aligned} \tag{3.4}$$

as claimed. □

Proposition 3.2.2. *The conditional distribution of M_k given J_k is equal to the distribution of $1+G$ where G is a geometric random variable with success probability $p = \alpha(J_k)$, i.e.*

$$\mathbf{P}[M_k = m|J_k] = (1-p)^{m-1}p = [1-\alpha(J_k)]^{m-1}\alpha(J_k), \text{ for } m = 1, 2, 3, \dots \tag{3.5}$$

and furthermore $\mathbb{E}[M_k \mid J_k] = 1/p = 1/\alpha(J_k)$.

Proof. If the original chain is at state x, then it has probability $p = \alpha(x)$ of leaving x on the next step, or probability $1-\alpha(x)$ of remaining at x. Hence, the probability that it will remain at x for m steps total (i.e., $m-1$ additional steps), and then leave at the next step, is equal to $(1-p)^{m-1}p$, as claimed. □

Proposition 3.2.3. *If the original chain $\mathbf{P}$ is irreducible, then so is the jump chain $\hat{\mathbf{P}}$*

Proof. Let $x, y \in S$. Since $\mathbf{P}$ is irreducible, there is a path $x = x_0, x_1, x_2, ..., x_m = y$ with $P(x_{i+1} \mid x_i) > 0$ for all i. Without loss of generality, we can assume the $\{x_i\}$ are all distinct. But if $\mathbf{P}(x_{i+1} \mid x_i) > 0$, then Proposition 3.2.1 implies that also $\hat{\mathbf{P}}(x_{i+1} \mid x_i) > 0$. Hence, $\hat{\mathbf{P}}$ is also irreducible. □

Proposition 3.2.4. *If the original chain $\mathbf{P}$ has stationary distribution π, then the jump chain $\hat{\mathbf{P}}$ has stationary distribution $\hat{\pi}$ given by $\hat{\pi}(x) = c\alpha(x)\pi(x)$ where $c = \Big(\sum_y \alpha(y)\pi(y)\Big)^{-1}$.*

Proof. Recall that on a discrete space, π is stationary for $\mathbf{P}$ if and only if $\sum_x \pi(x)\mathbf{P}(y \mid x) = \pi(y)$ for all $y\mathcal{S}$. In that case, we compute that

$$\begin{aligned}
\sum_x \hat{\pi}(x)\hat{\mathbf{P}}(y \mid x) &= \sum_x \Big(c\alpha(x)\pi(x)\Big)\Big(\mathbf{P}(y \mid x)/\alpha(x)\mathbf{1}_{y\neq x}\Big) \\
&= c\sum_{x\neq y} \Big(\pi(x)\mathbf{P}(y \mid x)\Big) \\
&= c\sum_x \Big(\pi(x)\mathbf{P}(y \mid x)\Big) - \pi(y)P(y \mid y) \\
&= c\pi(y) - \pi(y)\mathbf{P}(y \mid y) \\
&= c\pi(y)\Big(1 - \mathbf{P}(y \mid y)\Big) \\
&= c\pi(y)\alpha(y) \\
&= \hat{\pi}(y)
\end{aligned} \tag{3.6}$$

so that $\hat{\pi}$ is stationary for $\hat{\mathbf{P}}$, as claimed. □

Remark 3.2.5. *Most of the results presented in this section are already known in the Metropolis-Hastings context: the geometric distribution of the holding times is noted in Lemma 1(3) of [19] and Proposition 1(a) of [45]; the modified transition probabilities of the jump chain are stated in Proposition 1(b) of [45]; and the relationship between the stationary distributions of the original and jump chains is used in Lemma 1(4) of [19], Proposition 1(c) of [45], and Proposition 2.1 of [54], Lemma 1 of [20], and Section 2 of [18].*

Remark 3.2.6. *It is common that simple modifications of reversible chains lead to simple modifications of their stationary distributions. For example, if a reversible chain is restricted to a subset of the state space, so any moves out of the subset are rejected with the chain staying where it is, then its stationary distribution is equal to the original stationary distribution conditional on being in that subset since the detailed balance equation still holds on the subset. However, that property does not hold without reversibility. For a simple counter-example, let* $\mathcal{S} = \{A, B, C\}$, *with* $\mathbf{P}(B|A) = \mathbf{P}(C|B) = \mathbf{P}(A|C) = 3/4$, *and* $\mathbf{P}(C|A) = \mathbf{P}(A|B) = \mathbf{P}(B|C) = 1/4$. *Then if* $\mathcal{S}_0 = A, B$, *then the stationary distribution of the original chain is* $(1/3, 1/3, 1/3)$, *but the stationary distribution of the chain restricted to* $\mathcal{S}_0$ *is* $(1/4, 3/4)$. *We were thus surprised that Proposition 3.2.4 holds even for non-reversible chains.*

3.3 Using the Jump Chain for Estimation

The Jump Chain can be used for estimation, as we now discuss. This approach has also been taken by others; see Remarks 9 and 14 below.

Theorem 3.3.1. *Given an irreducible Markov chain* $\{X_n\}$ *with transition probabilities* $\mathbf{P}(y \mid x)$ *and stationary distribution* π *on a state space* $\mathcal{S}$, *and a function* $h : \mathcal{S} \to \mathbb{R}$, *suppose we simulate the jump chain* $\{J_k\}$ *with the transition probabilities as in Proposition 3.2.1, and then simulate the multiplicities list* $\{M_k\}$ *from the conditional probabilities as in Proposition 3.2.2, and set*

$$\bar{e}_K = \frac{\sum_{k=1}^{K} M_k h(J_k)}{\sum_{k=1}^{K} M_k} \tag{3.7}$$

Then $\bar{e}_K$ *is a consistent estimator of the expected value* $\mathbb{E}_\pi(h)$, *i.e.* $\lim_{K\to\infty} \bar{e}_K = \mathbb{E}_\pi(h)$ *with probability* 1.

Proof. Recall (e.g. [61]) that the usual estimator $\hat{e}_N = \sum_{n=1}^{N} h(X_n)$ is consistent. That is, $\lim_{K\to\infty} \hat{e}_N = \mathbb{E}_\pi(h)$ with probability 1. Then, it is seen that $\bar{e}_K = \frac{\sum_{k=1}^{K} M_k h(J_k)}{\sum_{k=1}^{K} M_k} = \hat{e}_{\sum_{k=1}^{K} M_k} = \hat{e}_{N(K)}$ where $N(K) = \sum_{k=1}^{K} M_k$. Since each $M_k \geq 1$, $\lim_{K\to\infty} \hat{e}_N = \mathbb{E}_\pi(h)$, so $\lim_{K\to\infty} \bar{e}_K = \lim_{K\to\infty} \hat{e}_{N(K)} = \lim_{K\to\infty} \hat{e}_N \mathbb{E}_\pi(h)$ with probability 1, as claimed. □

Remark 3.3.2. *The consistency of the estimate 3.7, and similarly those of Theorems 12 and 13 below, is already known in the Metropolis-Hastings context; see equation (3) of [54], Section 2 of [19], and equation (2) of [45].*

On the other hand, combining the Markov chain Law of Large Numbers with Proposition 3.2.3 and Proposition 3.2.4 immediately gives:

Proposition 3.3.3. *Under the above assumptions, if we simulate the jump chain* $\{J_k\}$ *with the transition probabilities* $\hat{P}$, *then for any function* $g : \mathcal{S} \to R$ *with* $\hat{\pi}|g| < \infty$, *we have*

$$\lim_{k\to\infty} \frac{1}{K} \sum_{k=0}^{K} g(J_k) = \hat{\pi}(g) := \sum_{x\in\mathcal{S}} g(x)\hat{\pi}(x) = c \sum_{x\in\mathcal{S}} g(x)\alpha(x)\pi(x), \text{ with probability 1} \tag{3.8}$$

Corollary 3.3.4. *Under the above assumptions, if we simulate the jump chain $\{J_k\}$ with the transition probabilities $\hat{P}$, then for any function $h : \mathcal{S} \to R$ with $\hat{\pi}|h| < \infty$, we have*

$$\lim_{k\to\infty} \frac{1}{cK} \sum_{k=0}^{K} \frac{h(J_k)}{\alpha(J_k)} = \pi(h) = \sum_{x\in\mathcal{S}} h(x)\pi(x), \text{ with probability 1} \tag{3.9}$$

Proof. Let $g(x) = \frac{h(x)}{c\alpha(x)}$.Then since $\hat{\pi}|h| < \infty$, we have

$$\begin{aligned}
\hat{\pi}|g| &= \sum_{x\in\mathcal{S}} |g(x)\times|\hat{\pi}(x) \\
&= \sum_{x\in\mathcal{S}} \frac{|h(x)|}{c\alpha(x)} \times c\alpha(x)\pi(x) \\
&= \sum_{x\in\mathcal{S}} \frac{|h(x)|}{c\alpha(x)} \times c\alpha(x)\pi(x) \\
&= \sum_{x\in\mathcal{S}} |h(x)| \times \pi(x) \\
&= \pi|h| < \infty
\end{aligned} \tag{3.10}$$

So, the result follows upon plugging this g into Proposition 3.3.3. □

Theorem 3.3.5. *Under the above assumptions, if we simulate the jump chain $\{J_k\}$ with the transition probabilities $\hat{P}$, then for any function $h : \mathcal{S} \to \mathbb{R}$ with $\hat{\pi}|h| < \infty$, we have*

$$\lim_{K\in\infty} \frac{\sum_{k=1}^{K}[h(J_k)/\alpha(J_k)]}{\sum_{k=1}^{K}[1/\alpha(J_k)]} = \pi(h), \text{ with probability 1.} \tag{3.11}$$

Proof. Setting $h \equiv 1$ in Corollary 3.3.4 gives that with probability 1, $\lim_{k\to\infty} \frac{1}{cK} \sum_{k=0}^{K} \frac{1}{\alpha(J_k)} = 1$. We then compute that

$$\begin{aligned}
\lim_{K\in\infty} \frac{\sum_{k=1}^{K}[h(J_k)/\alpha(J_k)]}{\sum_{k=1}^{K}[1/\alpha(J_k)]} &= \lim_{K\in\infty} \frac{\frac{1}{cK}\sum_{k=1}^{K}[h(J_k)/\alpha(J_k)]}{\frac{1}{cK}\sum_{k=1}^{K}[1/\alpha(J_k)]} \\
&= \lim_{K\in\infty} \frac{\sum_{x\in\mathcal{S}} h(x)\pi(x)}{1} \\
&= \pi(h)
\end{aligned} \tag{3.12}$$

as claimed. □

Comparing Theorems 3.3.1 and 3.3.5, we see that they coincide except that each multiplicity random variable M_k has been replaced by its mean $1/\alpha(J_k)$. Proposition 3.2.2. Finally, we note that although our computer hardware does not allow us to exploit it, most of the above carries over to Markov chains with densities on general state spaces, as follows. Note that the proofs are very similar to the discrete case and are thus omitted.

Theorem 3.3.6. *Let $\mathcal{S}$ be a general state case, and μ be a σ-finite reference measure on $\mathcal{S}$. Suppose a Markov chain on $\mathcal{S}$ has transition probabilities $\mathbf{P}(x, dy) \propto q(x, y)\mu(dy)$ for $q : \mathcal{S} \times \mathcal{S} \to [0, 1]$. Again let $\hat{P}$ be the transitions for the corresponding jump chain J_k with multiplicities M_k. Then:*

1. *$\hat{P}(x,\{x\}) = 0$, and for $x \neq y$, $\hat{\mathbf{P}}(x,dy) = \frac{q(x,y)}{\int q(x,z)\mu(dz)}\mu(dy)$*

2. *The conditional distribution of M_k given J_k is equal to the distribution of $1+G$ where G is a geometric random variable with success probability $p = \alpha(J_k)$ where $\alpha(x) = \mathbf{P}[X_{k+1} \neq x \mid X_k = x] = \int q(x,z)\mu(dz) = 1 - r(x) = 1 - \mathbf{P}(x \mid x)$*

3. *If the original chain is ϕ-irreducible (see, e.g., [61]) for some positive σ-finite measure ϕ on $\mathcal{X}$, then the jump chain is also ϕ-irreducible for the same ϕ.*

4. *If the original chain has stationary distribution $\pi(x)\mu(dx)$, then the jump chain has stationary distribution given by $\hat{\pi}(x) = c\alpha(x)\pi(x)\mu(dx)$ where $c^{-1} = \int \alpha(y)\pi(y)\mu(dy)$*

5. *If $h : \mathcal{S} \to \mathbb{R}$ has finite expectation, then with probability 1,*

$$\lim_{K\to\infty} \frac{\sum_{k=1}^{K} M_k h(J_k)}{\sum_{k=1}^{K} M_k} = \lim_{K\to\infty} \frac{\sum_{k=1}^{K} [\frac{h(J_k)}{\alpha(J_k)}]}{\sum_{k=1}^{K} [\frac{1}{\alpha(J_k)}]} = \pi(h) := \int h(x)\pi(x)\mu(dx)$$

3.4 Application to the Metropolis-Hastings Algorithm

Suppose now that the original chain $\{X_n\}$ is a Metropolis-Hastings algorithm, with proposal probabilities $\mathcal{Q}(x,y)$. Then for $x \neq y$, $\mathbf{P}(y \mid x) = \mathcal{Q}(x,y)\min\{1, \frac{\pi(y)\mathcal{Q}(y,x)}{\pi(x)\mathcal{Q}(x,y)}\}$. Hence, by Proposition 3.2.1, the jump chain transition probabilities have $\hat{\mathbf{P}}(x \mid x) = 0$ and for $x \neq y$ are given by

$$\begin{aligned} \hat{\mathbf{P}}(y \mid x) &\coloneqq \mathbf{P}[J_1 = y \mid J_0 = x] \\ &= \frac{\mathcal{Q}(x,y)\min\left\{1, \frac{\pi(y)\mathcal{Q}(y,x)}{\pi(x)\mathcal{Q}(x,y)}\right\}}{\sum_{z\neq x}\mathcal{Q}(x,z)\min\left\{1, \frac{\pi(z)\mathcal{Q}(z,x)}{\pi(x)\mathcal{Q}(x,z)}\right\}}. \end{aligned} \tag{3.13}$$

Also, here

$$\alpha(x) = \sum_{y\neq x}\mathbf{P}(y \mid x) = \sum_{z\neq x}\mathcal{Q}(x,z)\min\left\{1, \frac{\pi(z)\mathcal{Q}(z,x)}{\pi(x)\mathcal{Q}(x,z)}\right\}. \tag{3.14}$$

A special case is where the proposal probabilities $\mathcal{Q}(x,\cdot)$ are uniform over all neighbors of x, where each state has the same number N of neighbors. We assume that x is not a neighbor of itself and that x is a neighbor of y if and only if y is a neighbor of x. Then for $x \neq y$, $\mathbf{P}(y \mid x) = \frac{1}{N}\min\{1, \frac{\pi(y)}{\pi(x)}\}$. And, by Proposition 3.2.1, the jump chain transition probabilities have $\hat{\mathbf{P}}(x \mid x) = 0$ and for $x \neq y$ are given by

$$\hat{\mathbf{P}}(y \mid x) = \frac{\min\left\{1, \frac{\pi(y)}{\pi(x)}\right\}}{\sum_{z\in\mathcal{N}(x)}\min\left\{1, \frac{\pi(z)}{\pi(x)}\right\}}. \tag{3.15}$$

Note that, $\mathcal{N}(x)$ represents the set of all neighbors of x. Also, here

$$\alpha(x) = \frac{1}{N}\sum_{z\neq x}\min\left\{1, \frac{\pi(z)}{\pi(x)}\right\}. \tag{3.16}$$

The use of the Theorems 3.3.1 and 3.3.5 in the context of uniform Metropolis algorithms can be carried out very efficiently using special parallelized computer hardware, which will be discussed in Section 3.7, and was our original motivation for this investigation.

Remark 3.4.1. *The "n-fold way" of [10] considers the Ising model and selects the next site to flip proportional to its probability of flipping, by first classifying all sites in terms of their spin and neighbor counts. This creates a rejection-free Metropolis-Hastings algorithm in the same spirit as our approach, though specific to the Ising model. Later authors parallelized their algorithm, still for the Ising model; see e.g. [52] and [49].*

Given the above properties of the jump chain, the Rejection-Free algorithm is a sampling method that produces the jump chain as described by Algorithm 4. Note that the Rejection-Free algorithm described here can only deal with the discrete cases with at most a finite number of neighbors for all states.

Algorithm 4 Rejection-Free algorithm for discrete case

initialize J_0

for k in 1 to K **do**

calculate multiplicity list $M_{k-1} \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}(J_{k-1})} \mathcal{Q}(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, z)}\right\}$$

choose the next jump chain State $J_k \in \mathcal{N}(J_{k-1})$ such that

$$\hat{\mathbf{P}}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, y)}\right\}$$

end for

Note that, in Algorithm 4, when we need to pick our next state J_k according to the given probabilities, we can use the technique shown in Section 3.8, which is designed to sample proportionally by parallel hardware. In addition, even when the Rejection-Free is applied to a single core implementation, such a technique is still faster than other methods to sample proportionally, since calculating arg min is much faster than calculating the summation.

Again, Algorithm 4 ensures (assuming irreducibility) that the expected value $\mathbb{E}_\pi(h)$ of a functional $h : \mathcal{S} \to \mathbb{R}$ with respect to π can be estimated by $\frac{\sum_{k=1}^K M_k\, h(J_k)}{\sum_{k=1}^K M_k}$ for sufficiently large run length K, while avoiding any rejections. Rejection-Free can lead to great speedup in examples where rejections frequently happen for the Metropolis-Hastings algorithm.

3.5 Hybrid Chains

Sometimes we have two or more different Markov chains and we wish to alternate between them in some pattern to form a hybrid Markov chain. We reviewed the hybrid chains for regular MCMC in Section 2.3.1. In addition, we might wish to use rejection-free sampling for some or all of the individual chains. However, if this is done naively, it can lead to bias:

Example 9. *Let $\mathcal{S} = \{1, 2, 3, 4\}$, and $\pi = (1 - \epsilon, 3\epsilon, 1 - \epsilon, 1 - \epsilon)/3$ for some small positive number ϵ. For example, $\epsilon = 0.001$. Let $\mathcal{Q}_1(x, x+1) = \mathcal{Q}_1(x, x-1) = 1/2$ and $\mathcal{Q}_2(x, x+1) = \mathcal{Q}_2(x, x+2) = \mathcal{Q}_2(x, x-1) = \mathcal{Q}_2(x, x-2) = 1/4$ be two different proposal kernels, and let $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ be usual Metropolis algorithms for π with proposals $\mathcal{Q}_1$ and $\mathcal{Q}_2$ respectively. Then, each of $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ will*

converge to π, as will the algorithm of alternating between $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ any fixed number of times. However, if we instead alternate between doing one jump step of $\boldsymbol{p}_1$ and then one jump step of $\boldsymbol{p}_2$, then this combined chain will not converge to the correct distribution. Indeed, the corresponding escape probabilities $\alpha_1(x)$ and $\alpha_2(x)$ are all reasonably large (at least 1/4) except for $\alpha_1(1) = \epsilon/2$ which is extremely small. This means that when our algorithm uses $\boldsymbol{p}_1$ from state 1 then it will have an extremely large multiplicity M_k which will lead to an extremely large weight of the state A. Indeed, if we use the alternating jump chains algorithm, then the estimators $\bar{e}_K$ as in Theorem 3.3.1 will have the property that as $\epsilon \downarrow 0$, their limiting value converges to $h(1)$ instead of $\pi(h)$, i.e.

$$\lim_{\epsilon \downarrow 0} \lim_{K \to \infty} \bar{e}_K = h(1). \tag{3.17}$$

Hence, convergence to π fails in this case.

Thus, if we proceed with alternating chains naively for Rejection-Free, it can lead to bias. For each proposal distribution $\mathcal{Q}^i$, we need to get the same amount of samples by the original sample size ($\sum_{k=1}^{K} M_k$) instead of the jump sample size (K) to fix the bias problem. For $\mathcal{I}$ proposal distributions $\mathcal{Q}^0, \mathcal{Q}^1, \ldots, \mathcal{Q}^{\mathcal{I}-1}$, the corresponding neighbor sets are $\mathcal{N}^0, \mathcal{N}^1, \ldots, \mathcal{N}^{\mathcal{I}-1}$ where $\mathcal{N}^i(x) = \{y : y \in \mathcal{S}, \mathcal{Q}^i(x, y) > 0\}$ for $i = 0, 1, \ldots, \mathcal{I} - 1$. Then, if we choose to switch between proposal distributions for L_0 original samples, we can do alternating chains in a Rejection-Free manner as Algorithm 5. Again, note that we used superscripts here for the proposal distribution $\mathcal{Q}^i$ and neighbor sets $\mathcal{N}^i$ for Rejection-Free here as the index for different proposals and neighbor sets. Later for Partial Neighbor Search, we will be using proposal distributions with subscripts like $\mathcal{Q}_i$, and we will discuss their difference later in Section 4.2.1.

Algorithm 5 Rejection Free algorithm with Alternating Chains

initialize $i \leftarrow 0$ ▷ start with proposal distribution $\mathcal{Q}^0$
initialize $L \leftarrow L_0$ ▷ start with L_0 remaining original samples
initialize J_0
for k in 1 to K **do**
  calculate multiplicity list $m \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}^i(J_{k-1})} \mathcal{Q}^i(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}^i(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}^i(J_{k-1}, z)}\right\}$$

  if $m \leq L$ **then** ▷ if we have enough remaining original samples
    $M_{k-1} \leftarrow m$, $L \leftarrow L - m$
    choose the next jump chain State $J_k \in \mathcal{N}^i(J_{k-1})$ such that

$$\hat{\mathbf{P}}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}^i(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}^i(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}^i(J_{k-1}, y)}\right\}$$

  else ▷ if we don't have enough remaining original samples
    $M_{k-1} \leftarrow L$, $L \leftarrow L_0$, $J_k \leftarrow J_{k-1}$, $i \leftarrow (i + 1 \mod \mathcal{I})$
    ▷ stay at J_{k-1} for L times and switch to the next $\mathcal{N}^i$
  end if
end for

Algorithm 5 is equivalent to Algorithm 2 except that algorithm 5 computes immediate repeated state for each proposal distribution all at once. As such, it has no bias, is consistent, and will

converge to the target distribution correctly. Again, in Algorithm 5, when we need to pick our next state J_k according to the given probabilities, we can use the technique shown in Section 3.8.

3.6 Application to Parallel Tempering

Parallel tempering [32], [79] proceeds by considering different versions of the target distribution π powered by different inverse-temperatures β, of the form $\pi^{(\beta)} \propto \big(\pi(x)\big)^{\beta}$. It runs separate MCMC algorithms on each target $\pi^{(\beta)}$, for some fixed number of iterations, and then proposes to swap pairs of values $X^{(\beta_1)} \leftrightarrow X^{(\beta_2)}$. This swap proposal is accepted with the usual Metropolis algorithm probability

$$\min\left[1, \frac{\pi^{(\beta_1)}(X^{(\beta_2)})\pi^{(\beta_2)}(X^{(\beta_1)})}{\pi^{(\beta_1)}(X^{(\beta_1)})\pi^{(\beta_2)}(X^{(\beta_2)})}\right] \tag{3.18}$$

which preserves the product target measure $\Pi_\beta \pi^{(\beta)}$.

But suppose we instead want to run parallel tempering using jump chains, i.e. using a rejection-free algorithm within each temperature. If we run a fixed number of rejection-free moves of each within-temperature chain, followed by one "usual" swap move, then this can lead to bias, as the following example shows.

Example 10. *Let $\mathcal{S} = \{1, 2, 3\}$, with $\pi(1) = \pi(3) = 1/4$ and $\pi(2) = 1/2$. Suppose there are just two inverse-temperature values, $\beta_0 = 1$ and $\beta_1 = 5$. Suppose each within the temperature chain proceeds as a Metropolis algorithm, with proposal distribution given by $\mathcal{Q}(x, y) = 1/2$ whenever $y \neq x$. (That is, we can regard the three states of $\mathcal{S}$ as being in a circle, and the chain proposes to move one step clockwise or counter-clockwise with probability 1/2 each, and then accepts or rejects this move according to the usual Metropolis procedure.) If we run a usual parallel tempering algorithm, then the within-temperature moves will converge to the corresponding stationary distributions $\pi^{(1)} = \pi = (1/4, 1/2, 1/4)$ and $\pi^{(5)} = (1/34, 32/34, 1/34)$ respectively. Then, given current chain values $X^{(1)}$ and $X^{(5)}$, if we attempt a usual swap move, it will be accepted with a probability*

$$\min\left[1, \frac{\pi^{(1)}(X^{(5)})\pi^{(5)}(X^{(1)})}{\pi^{(1)}(X^{(1)})\pi^{(5)}(X^{(5)})}\right] \tag{3.19}$$

These steps will all preserve the product stationary distribution $\pi^{(1)} \times \pi^{(5)}$, as they should. However, if we instead run a rejection-free within-temperature chain, then convergence fails.

Indeed, from each state, the jump chain is equally likely to move to either of the other two states, so each jump chain will converge to the uniform distribution on $\mathcal{S}$. The acceptance probability as in Equation 3.19 will then lead to incorrect distributional convergence, e.g. if $X^{(1)} = 2$ and $X^{(5)} = 3$, then a proposal to swap $X^{(1)}$ and $X^{(5)}$ will always be accepted, leading to an excessively large probability that $X^{(1)} = 3$. Indeed, if we run a simple simulation in R the fraction of time that $X^{(1)} = 3$ right after a swap proposal is about 44%, much larger than the 1/3 probability it should be.

To get rejection-free parallel tempering to converge correctly, we recall from Proposition 3.3.3 that the rejection-free chains actually converge to the modified stationary distributions $\hat{\pi}$, not π.

We should thus modify the acceptance probability in Equation 3.19 to:

$$\begin{aligned} &\min\left[1, \frac{\hat{\pi}^{(\beta_1)}(X^{(\beta_2)})\hat{\pi}^{(\beta_2)}(X^{(\beta_1)})}{\hat{\pi}^{(\beta_1)}(X^{(\beta_1)})\hat{\pi}^{(\beta_2)}(X^{(\beta_2)})}\right] \\ =&\min\left[1, \frac{\alpha^{(\beta_1)}(X^{(\beta_2)})\pi^{(\beta_1)}(X^{(\beta_2)})\alpha^{(\beta_2)}(X^{(\beta_1)})\pi^{(\beta_2)}(X^{(\beta_1)})}{\alpha^{(\beta_1)}(X^{(\beta_1)})\pi^{(\beta_1)}(X^{(\beta_1)})\alpha^{(\beta_2)}(X^{(\beta_2)})\pi^{(\beta_2)}(X^{(\beta_2)})}\right] \end{aligned} \tag{3.20}$$

Such swaps will preserve the product modified stationary distribution $\Pi_\beta \hat{\pi}^{(\beta)}$, rather than trying to preserve the unmodified stationary distribution $\Pi_\beta \pi^{(\beta)}$. Note that, if necessary, the escape probabilities $\alpha(x)$ can be estimated from a preliminary run. The rejection-free parallel tempering algorithm will thus converge to $\Pi_\beta \hat{\pi}^{(\beta)}$, thus still allowing for valid inference as in Theorems 3.3.1 and 3.3.5.

Example 10 (continuing from p. 31)**.** *In this example,* $\alpha^{(1)}(A) = \alpha(1)(C) = \alpha^{(5)}(A) = \alpha(5)(C) = 1$, $\alpha^{(1)}(B) = 1/2$, *and* $\alpha^{(5)}(B) = 1/32$. *So, if* $X^{(1)} = 2$ *and* $X^{(5)} = 3$, *then according to Equation 3.20, a proposal to swap* $X^{(1)}$ *and* $X^{(5)}$ *will be accepted with probability*

$$\begin{aligned} &\min\left[1, \frac{\alpha^{(1)}(X^{(5)})\pi^{(1)}(X^{(5)})\alpha^{(5)}(X^{(1)})\pi^{(5)}(X^{(1)})}{\alpha^{(1)}(X^{(1)})\pi^{(1)}(X^{(1)})\alpha^{(5)}(X^{(5)})\pi^{(5)}(X^{(5)})}\right] \\ =&\min\left[1, \frac{(1)(1/4)(1/32)(1/2)}{(1/2)(1/2)(1)(1/34)}\right] \\ =&\min\left[1, \frac{17}{32}\right] \\ =&\frac{17}{32} \end{aligned} \tag{3.21}$$

and such swaps will instead preserve the product stationary distribution $\hat{\pi}^{(1)} \times \hat{\pi}^{(5)}$. *Indeed, if we run this simulation in R, the fraction of time that* $X^{(1)} = 3$ *right after a swap proposal with this modified acceptance probability becomes about* $1/3$, *as it should be.*

3.7 Numerical Examples

In this section, we introduce applications and simulations to illustrate the advantage of Rejection-Free algorithm. We compare the efficiency of the Rejection-Free and standard Metropolis algorithms in three different examples. The first example is a Bayesian inference model on a real data set taken from the Education Longitudinal Study of [11]. The second example involves sampling from a two-dimensional ferromagnetic 4×4 Ising model. The third example is a pseudo-marginal [4] version of the Ising model. All three simulations show that the introduction of the Rejection-Free method leads to significant speedup. This provides concrete numerical evidence for the efficiency of using the rejection-free approach to improve the convergence to stationarity of the algorithms.

3.7.1 A Bayesian Inference Problem with Real Data

For our first example, we consider the Education Longitudinal Study [11] real data set consisting of final course grades of over $9,000$ students. We take a random subset of 200 of these $9,000$ students

	ESS per Iteration	ESS per second
Metropolis	0.0074	47
Rejection-Free	0.9100	4,261
Ratio	123.0	90.7

Table 3.1: Median of Effective Sample Sizes from 100 Runs each of the Metropolis and Rejection-Free algorithms

and denote their scores as $x_1, x_2 \ldots, x_{200}$. (Note that all scores in this data set are integers between 0 and 100.)

Our parameter of interest θ is the true average value of the final grades for these 200 students, rounded to 1 decimal place, so $\theta \in \{0.1, 0.2, 0.3, \ldots, 99.7, 99.8, 99.9\}$ is still discrete and can be studied using specialized computer hardware. The likelihood function for the grade x of a single student is the binomial distribution

$$\mathcal{L}(x \mid \theta) = \binom{100}{x} \left(\frac{\theta}{100}\right)^x \left(\frac{1-\theta}{100}\right)^{100-x}. \tag{3.22}$$

For our prior distribution, we take

$$\theta \sim \text{Unif}\{0.1, 0.2, 0.3, \ldots, 99.7, 99.8, 99.9\} \tag{3.23}$$

The posterior distribution $\pi(\theta)$ is then proportional to the prior probability function as Equation 3.23 times the likelihood function as Equation 3.22. Note that, this is a simple model used to illustrate how Rejection-Free works, and this model is somehow unrealistic since we assumed a common underlying parameter θ for all students.

We ran an Independence Sampler for this posterior distribution, with a fixed proposal distribution equal to the prior as Equation 3.23, either with or without the Rejection-Free modification. For each of these two algorithms, we calculated the effective sample size, defined as

$$ESS(\theta) = \frac{N}{1 + 2\sum_{k=1}^{\infty} \rho_k(\theta)} \tag{3.24}$$

where N is the number of posterior samples, $\rho_k(\theta)$ represents autocorrelation at lag k for the posterior samples of θ. For an MCMC chain with finite length, the sum $\sum_{k=1}^{\infty} \rho_k(\theta)$ cannot be taken over all k, so instead we just sum until the values of $\rho_k(\theta)$ become negligible. In Addition, to calculate the autocorrelation for the Rejection-Free chain, we have to convert it back to the original chain. For a fair comparison, we consider both the ESS per iteration and the ESS per second of CPU time.

Table 3.1 presents the median ESS per iteration, and median ESS per second, from 100 runs of 100,000 iterations each, for each of the two algorithms. We see from Table 3.1 that Rejection-Free outperforms the Metropolis algorithm by a factor of approximately one hundred, in terms of both ESS per iteration and ESS per second. This clearly illustrates the efficiency of the Rejection-Free algorithm.

3.7.2 Apply Rejection-Free to the Ising model

We next present a simulation study of a ferromagnetic Ising model on a two-dimensional 4×4 square lattice. The energy function for this model is given by

$$E(S) = -\sum_{i<j} J_{i,j} s_i s_j \tag{3.25}$$

where each spin $s_i, s_j \in \{-1, 1\}$, and $J_{i,j}$ represents the interaction between the i-th and j-th spins. To make only the neighboring spins in the lattice interact with each other, we take $J_{i,j} = 1$ for all neighbors i and j, and $J_{i,j} = 0$ otherwise. The Ising model then has probability distribution proportional to the exponential of the energy function:

$$\pi(S) \propto \exp\{-E(S)\}. \tag{3.26}$$

We investigate the efficiency of the samples produced in four different scenarios: Metropolis algorithm and Rejection-Free, both with and without Parallel Tempering. For the Parallel Tempering versions, we set

$$\pi_T(S) \propto \exp\{-\frac{E(S)}{T}\}. \tag{3.27}$$

Here $T = 1$ is the temperature of interest (which we want to sample from). We take $T = 2$ as the highest temperature, since when $T = 2$ the probability distribution for magnetization is quite flat (with highest probability $\mathbf{P}[M(S) = 14] = 0.083$, and lowest probability $\mathbf{P}[M(S) = 2] = 0.037$). Including the one additional temperature $T = \sqrt{2}$ gives three temperatures $T \in \{1, \sqrt{2}, 2\}$ in geometric progression, with an average swap acceptance rate 31.6% which is already higher than the 23.4% recommended in [67], indicating that three temperatures are enough.

In this example, we measure the distance to stationarity by the total variation distance. Given the Markov chain $\{X_k\}_{k=1}^K$ generated by the Metropolis algorithm, the sampling distribution is defined as $\mathcal{P}_{\text{Sampled}}(x) = \frac{\sum_{k=1}^K \mathbb{1}(X_k = x)}{K}$, $\forall x \in \mathcal{S}$, where $\mathbb{1}$ represents the indicator function. In addition, for the jump chain $\{J_k, M_k\}_{k=1}^K$ generated by Rejection-Free, the sampling distribution is defined as $\mathcal{P}_{\text{Sampled}}(x) = \frac{\sum_{k=1}^K M_k \times \mathbb{1}(J_k = x)}{\sum_{k=1}^K M_k}$, $\forall x \in \mathcal{S}$. The corresponding total variation distance (TVD) in both cases is defined as

$$\text{TVD}(\mathcal{P}_{\text{Sampled}}, \pi) = \frac{1}{2} \sum_{x \in \mathcal{S}} \Big| \mathcal{P}_{\text{Sampled}}(x) - \pi(x) \Big|. \tag{3.28}$$

According to the definition, TVD is strictly between $[0, 1]$. When the sampling distribution $\mathcal{P}_{\text{Sampled}}$ gets closer to the target distribution π, TVD will decrease to 0. In other words, convergence to stationarity is described by how quickly TVD(n) decreases to 0.

In addition, we study the convergence of the magnetization value, where the magnetization of a given state S of the Ising model is defined as:

$$M(S) = \sum_{i=1}^{N} s_i \tag{3.29}$$

For our 4×4 Ising model,

$$M(S) \in \mathcal{M} = \{-16, -14, -12, \ldots, -2, 0, 2, \ldots, 12, 14, 16\} \tag{3.30}$$

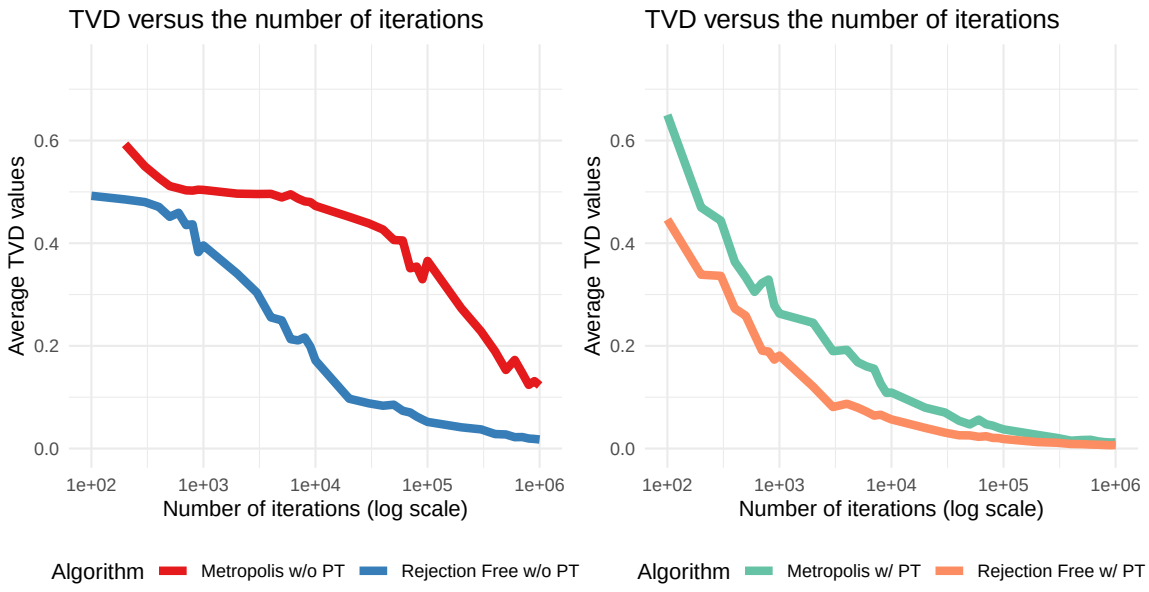

Figure 3.5: Average of total variation distance between sampled and actual distributions as a function of the number of iterations for four scenarios: Metropolis versus Rejection-Free without Parallel Tempering (left) and with Parallel Tempering (right)

Thus, the corresponding TVD between the sampled and the actual magnetization distributions after K iterations can also be written as

$$\mathrm{TVD}(K) = \frac{1}{2} \sum_{m \in \mathcal{M}} |\mathbf{P}(M(X_k) = m) - \pi\{S : M(S) = m\}| \tag{3.31}$$

where $M(X_k)$ is the magnetization of the chain at iteration k, and $\pi\{S : M(S) = m\}$ represents the stationary probability of magnetization value m.

Figure 3.5 lists the average total variation distance TVD(n) for each version, as a function of the number of iterations n, based on 100 runs of each of the four scenarios, of 106 iterations each. It illustrates that, with or without Parallel Tempering, the use of the Rejection-Free provides significant speedup, and TVD decreases much more rapidly with the Rejection-Free method than without it. This provides concrete numerical evidence for the efficiency of using Rejection-Free to improve the convergence to stationarity of the algorithm.

We next consider the issue of computational cost. The Rejection-Free method requires computing probabilities for all neighbors of the current state. However, with specialized computer hardware, Rejection-Free can be very efficient since the calculation of the probabilities for all neighbors and the selection of the next state can both be done in parallel. The computational cost of each iteration of Rejection-Free is therefore equal to the maximum cost used on each neighbor. Similarly, for Parallel Tempering, we can calculate all of the different temperature chains in parallel. The average CPU time per iteration for each of the four different scenarios is presented in Table 3.2. It illustrates that the computational cost of Rejection-Free without Parallel Tempering was comparable to that of the usual Metropolis algorithm, though Rejection-Free with Parallel Tempering does require up to 50%

Algorithms	Average Time (nanoseconds)
Metropolis w/o PT	420
Rejection-Free w/o PT	407
Metropolis w/ PT	463
Rejection-Free w/ PT	611

Table 3.2: Average time used for each iteration of MCMC in four scenarios: Metropolis and Rejection-Free, both with and without Parallel Tempering

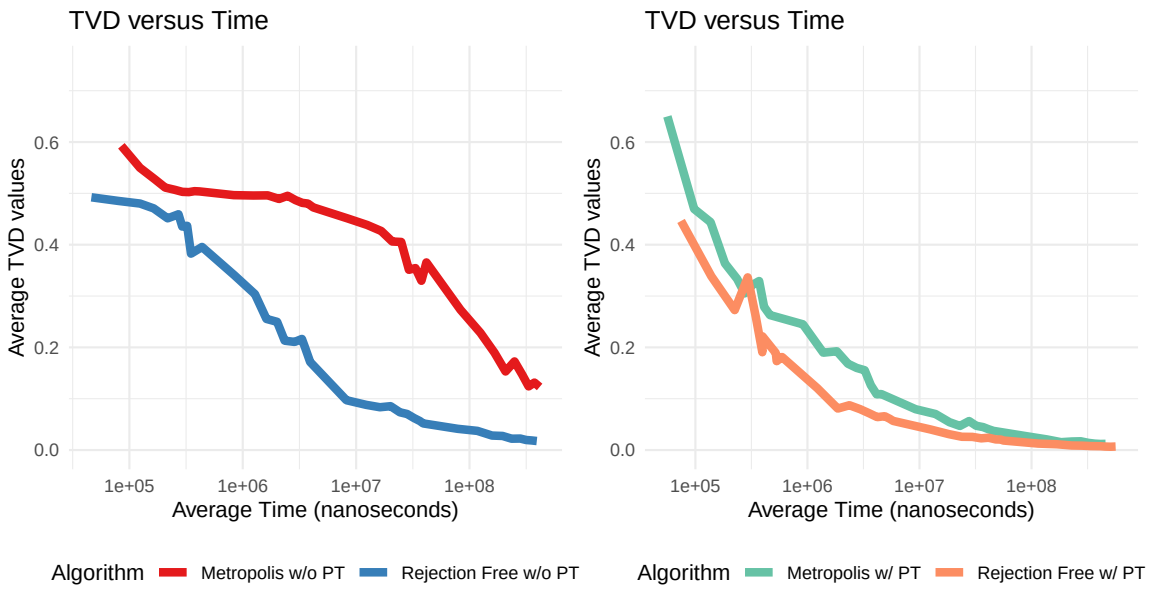

Figure 3.6: Average of total variation distance between sampled and actual distributions as a function of the average of time cost in four scenarios: Metropolis versus Rejection-Free without Parallel Tempering (left) and with Parallel Tempering (right)

more time than the other three scenarios.

Figure 3.6 shows the average total variation distance as a function of the total CPU time used for each algorithm. Figure 3.6 is quite similar to Figure 3.5 and gives the same overall conclusion: with or without Parallel Tempering, the use of Rejection-Free provides significant speedup, even when computational cost is taken into account.

As a final check, we also calculated the effective sample size, similar to the first example. First, we generated 100 MCMC chains of $100,000$ iterations each, from all four algorithms. Then, we calculated the effective sample size for each chain and normalized the results by either the number of iterations or the total CPU time for each algorithm. Table 3.3 shows the median of ESS per iteration and ESS per CPU second. It again illustrates that Rejection-Free can produce great speedups, increasing the ESS per CPU second by a factor of over 50 without Parallel Tempering, or a factor of 2 with Parallel Tempering.

	ESS per Iteration	ESS per second
Metropolis w/0 PT	0.0003	77.76
Rejection-Free w/o PT	0.0016	4227
Metropolis w/ PT	0.0057	11830
Rejection-Free w/ PT	0.0138	23459

Table 3.3: Median of Normalized Effective Sample Sizes for four scenarios: Metropolis and Rejection-Free, both with and without Parallel Tempering

3.7.3 Extend Rejection-Free to Pseudo-marginal Metropolis-Hastings

If the target density itself is not available analytically, but an unbiased estimate exists, then pseudo-marginal MCMC [4] can still be used to sample from the correct target distribution. We next apply the Rejection-Free method to a pseudo-marginal algorithm to show that Rejection-Free can provide speedups in that case, too.

In the previous example of the 4×4 Ising model, the target probability distributions were defined as

$$\pi(S) \propto \exp\{-\frac{E(S)}{T}\}. \tag{3.32}$$

We now pretend that this target density is not available, and we only have access to an unbiased estimator is given by

$$\pi_0(S) \propto \pi(S) \times A = \exp\{\frac{E(S)}{T}\} \times A \tag{3.33}$$

where $A \sim \text{Gamma}(\alpha = 10, \beta = 10)$ is a random variable (which is sampled independently every time we try to compute the target distribution). Note that $\mathbb{E}(a) = 10/10 = 1$, so $\mathbb{E}[\pi_0(S)] = \pi(S)$, and the estimator is unbiased (though A has variance $10/10^2 = 1/10 > 0$).

Using this unbiased estimate of the target distribution for pseudo-marginal MCMC, we again investigated the convergence of samples produced by the same four scenarios:

Metropolis and Rejection-Free, both with and without Parallel Tempering. Figure 3.7 shows the average total variation distance TVD(n) between the sampled and the actual magnetization distributions, for 100 chains, as a function of the iteration n, keeping all the other settings the same as before. This figure is quite similar to Figure 8, again showing that with or without Parallel Tempering, the use of Rejection-Free provides significant speedup, even in the pseudo-marginal case.

3.8 A Efficient Method to Sample Proportionally

Given $A_i > 0$, for $i = 1, 2, \ldots, N$, how can we sample Z so that $\mathbf{P}(Z = i) = \frac{A_i}{\sum_j A_j}$? We could choose $U \sim \text{Uniform}[0, 1]$, and then set $Z = \min\{i, \sum_{j=1}^{i} A_j > U \times \sum_{j=1}^{N} A_j\}$. However, this involves summing all of the A_j, which is inefficient. If $\sum_{j=1}^{N} A_j = 1$, then we could choose $U \sim \text{Uniform}[0, 1]$ and just set $Z = \min\{i, \sum_{j=1}^{i} A_j > U\}$, which is slightly easier and can be done by binary searching. However, it still requires summing lots of the A_j, which could still be inefficient. If $\sum_{j=1}^{N} A_j < 1$, then we could choose $U \sim \text{Uniform}[0, 1]$, and then still set $Z = \max\{i, \sum_{j=1}^{i} A_j > U\}$, except if no such i exists then we reject that choice of U and start again. In addition to the previous problems, this could involve lots of rejection if $\sum_{j=1}^{N} A_j$ is much smaller than 1, which is again inefficient. Another option is the following method, based on [24]; see also the n-fold way approach to kinetic Monte Carlo in [10].

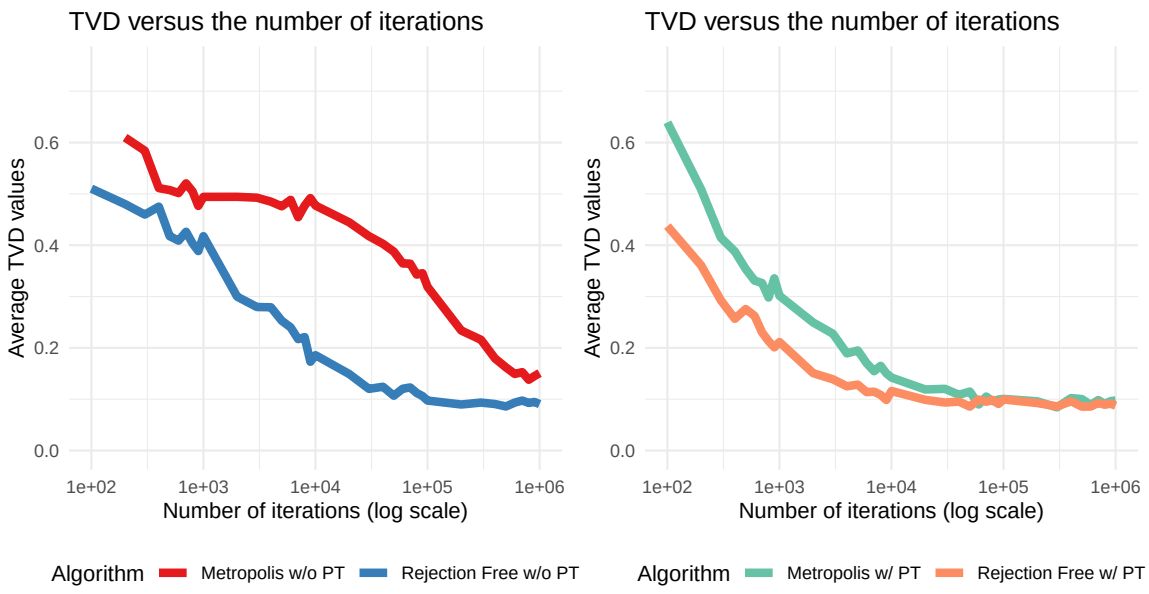

Figure 3.7: Average of total variation distance between sampled and actual distributions as a function of the number of iterations for probability with noise Gamma(10, 10) in four scenarios: Metropolis versus Rejection-Free without Parallel Tempering (left) and with Parallel Tempering (right)

Proposition 3.8.1. *Let $A_1, A_2, \ldots, A_N$ be positive numbers, Let $\{R_j\}_{j=1}^N$ be i.i.d. $\sim$ Uniform$[0,1]$, and let $d_j = -\frac{\log(R_j)}{A_j}$ for $j = 1, 2, \ldots, N$. Finally, set $Z = \arg\min_j d_j$. Then $\mathbf{P}[Z = i] = \frac{A_i}{\sum_j A_j}$, i.e. Z selects i from $\{1, 2, \ldots, N\}$ with probability proportional to A_i.*

Proof.

$$\begin{aligned}
\mathbf{P}[Z = i] &= \mathbf{P}[d_j > d_i, \forall j \neq i] \\
&= \mathbf{P}[-\frac{\log(R_j)}{A_j} > -\frac{\log(R_i)}{A_i}, \forall j \neq i] \\
&= \mathbf{P}[R_j < R_i^{A_j/A_i}, \forall j \neq i] \\
&= \int_0^1 \mathbf{P}[R_j < R_i^{A_j/A_i}, \forall j \neq i \mid R_i = x]dx \\
&= \int_0^1 \mathbf{P}[R_j < x^{A_j/A_i}, \forall j \neq i]dx \\
&= \int_0^1 \prod_{j \neq i} x^{A_j/A_i} dx \\
&= \int_0^1 x^{\sum_{j \neq i} A_j/A_i} dx \\
&= \frac{x^{[\sum_{j \neq i} A_j/A_i + 1]}}{\sum_{j \neq i} A_j/A_i + 1}\Bigg|_{x=0}^1 \\
&= \frac{A_i}{\sum_j A_j}
\end{aligned} \tag{3.34}$$

□

Proposition 3.8.1 is useful, especially when we apply Rejection-Free and PNS to parallel hardware. Each processor can calculate d_j for different j separately, and we only need to find the arg min to get a sample with probability proportional to the given values. On the other hand, for other methods of sampling, usually we have to calculate the summation from $\mathbf{P}[Z = i] = \frac{A_i}{\sum_j A_j}$ which is much slower than finding arg min. In addition, such a method is much faster even when we apply it to a single-core implementation.

Chapter 4

Sampling via Partial Neighbor Search

We talked about how to deal with the inefficiency of rejections from the Metropolis algorithm by the Rejection-Free algorithm. However, the Rejection-Free algorithm cannot be applied to all cases. For example, there is a limit to the number of parallel tasks that can be executed simultaneously on most specialized parallel hardware. The current version of Digital Annealing Unit (DAU) [57] can process the Rejection-Free algorithm efficiently up to 8,192 neighbors simultaneously with on-chip memory, and otherwise, Rejection-Free will be significantly slowed down due to the use of external memory. Accordingly, the ceiling on the number of neighbors that can be evaluated at each step can be a problem. Consequently, we present an enhanced version of the Rejection-Free called Partial Neighbor Search (PNS), which only considers part of the neighbors when applying the Rejection-Free technique, whereas the Rejection-Free technique means considering all selected neighbors and calculating the next state when ignoring any immediately repeated states.

4.1 Basic Partial Neighbor Search algorithm

In Algorithm 4, we can do this algorithm with parallel in computer hardware to produce more efficient samples. However, the number of tasks that can be computed simultaneously by the parallel hardware is not unlimited, while the number of neighbors $|\mathcal{N}(x)|$ can be super large. How can we take full advantage of the Rejection-Free with limited parallel hardware?

Assume the number of neighbors in Rejection-Free is at most N. That is, for $\forall x \in S$, $|\mathcal{N}(x)| \leq N$. In addition, assume the number of tasks that can be computed simultaneously by the parallel hardware is M. If $M > N$, then we can compute the transition probability of the original chain simultaneously by the parallel hardware, where the transition probability is

$$\mathbf{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, y)}\right\}. \tag{4.1}$$

Then the transition probability $\hat{\mathbf{P}}$ defined in Proposition 3.2.1 for the Rejection-Free algorithm as stated in Algorithm 4 is propositional to $\mathbf{P}$, $\forall y \neq J_{k-1}$. On the other hand, if $M \leq N$, the simplest way to take advantage of parallel hardware is to evenly distribute the calculation tasks of the

transition probabilities to each unit. In this case, each unit of parallel hardware needs to calculate the probabilities for either $\lfloor \frac{N}{M} \rfloor$ (the floor function) or $\lceil \frac{N}{M} \rceil$ (the ceiling function) times, and then we can put the information from all these parts together for the next step of the algorithm. This method works for processors designed for general purposes, such as Intel and AMD cores. However, as we introduced in Section 2.4, these chips are not specially designed for parallel computing, and off-chip communication significantly slows down the transfer rate of data to and from the cores [77]. Therefore, using Intel and AMD cores as parallel hardware is applicable but not ideal.

Moreover, several types of parallell hardware specialized for MCMC trials have been proposed. For example, the DAU we introduced in Section 2.4 is designed to minimize communication overhead in arithmetic circuitry and with memory. In addition, the dedicated processor provides a virtually Rejection-Free process, resulting in a throughput that is orders of magnitude faster than that of a general-purpose processor. The problem with this Fujitsu chip is that it is rigidly constrained by on-chip memory capacity relative to the problem size M that can be processed in parallel. For problem sizes $N > M$, it is impossible to compute transition probabilities for all neighborhoods to achieve Rejection-Free or similar parallel trials. The number of neighbors considered in each step must be limited to be within the on-chip memory capacity.

Intuitively, we can calculate everything the same as Rejection-Free while only considering some of the neighbors. The Basic Partial Neighbor Search algorithm (Basic PNS) is shown in Algorithm 6. Again, we focus on discrete cases with at most a finite number of neighbors here. We will talk about PNS for general state space in Section 4.4.

Algorithm 6 Basic Partial Neighbor Search algorithm

initialize J_0
for k in 1 to K **do**
pick the Partial Neighbor Set $\mathcal{N}_k(J_{k-1}) \subset \mathcal{N}(J_{k-1})$
choose the next jump chain State $J_k \in \mathcal{N}_k(J_{k-1})$ such that

$$\hat{\mathbf{P}}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, y)}\right\}$$

calculate multiplicity list $M_{k-1} \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}_k(J_{k-1})} \mathcal{Q}(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}(J_{k-1}, z)}\right\}$$

end for

Again, in Algorithm 6, when we need to pick our next state J_k according to the given probabilities, we can use the technique shown in Section 3.8. In addition, the only difference between the Basic PNS (Algorithm 6) and Rejection-Free (Algorithm 4) is that we only calculate the transition probability and all the corresponding values for a subset $\mathcal{N}_k$ of all the neighbors for each step within the loop. Here, $\mathcal{N}_k(J_{k-1})$ is a subset of $\mathcal{N}(J_{k-1})$ at our choice, and the subscript k in $\mathcal{N}_k$ represents the subset of neighbors for step k. For example, we can simply say that $\mathcal{N}_k(J_{k-1})$ is a random subset of $\mathcal{N}(J_{k-1})$ with half of the elements. In addition, $\mathcal{Q}_k(X, Y)$ is the corresponding proposal distribution satisfying $\mathcal{Q}_k(x, y) \propto \mathcal{Q}(x, y)$ for $Y \in \mathcal{N}_k(x)$ and $\mathcal{Q}_k(x, y) = 0$ otherwise. However, the Markov chain produced by Algorithm 6 is different from the true MCMC, and it might not converge to the true

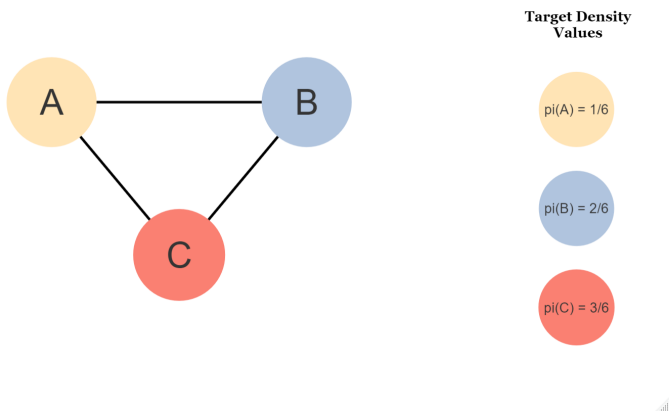

Figure 4.1: Diagram of Example 1 showing non-convergence property of the Basic PNS.

density π, as we now show.

Example 11. *The example illustrating the non-convergence property of Basic PNS is shown in Figure 4.1, from which we have $\pi(A) \propto 1$, $\pi(B) \propto 2$, and $\pi(C) \propto 3$. We consider the Basic PNS algorithm with a uniform proposal distribution $\mathcal{Q}$. In addition, only half of the neighbors are chosen for $\mathcal{N}_k$ at each step. That is, we only need to consider one neighbor each time.*

Then, if the MCMC is located at state A, then $\mathcal{N}(A) = \{B, C\}$. $\mathcal{N}_k(A) = \{B\}$ or $\{C\}$ each with 50% probability, and thus, the algorithm will force the chain to move to either B or C with 50% probability. Similarly, when the Markov chain is located at state B, the next state will be A or C with 50% probability, and when the Markov chain is located at state C, the next state will be A or B with 50% probability.

On the other hand, we can calculate the corresponding multiplicity lists M_A, M_B, and M_C at state A as follows:

1. $\hat{\mathbf{P}}[B \mid A] \propto \mathbf{P}[B \mid A] = \mathcal{Q}(A, B) \min\{1, \frac{\pi(B)\mathcal{Q}(B,A)}{\pi(A)\mathcal{Q}(A,B)}\} = 0.5$;
2. $\hat{\mathbf{P}}[C \mid A] \propto \mathbf{P}[C \mid A] = \mathcal{Q}(A, C) \min\{1, \frac{\pi(C)\mathcal{Q}(C,A)}{\pi(A)\mathcal{Q}(A,C)}\} = 0.5$;
3. *the transition probabilities $\hat{\mathbf{P}}$ from A to either B or C in Rejection-Free are both 50%;*
4. $M_A = 1 + G$ *where* $G \sim Geom(\mathbf{P}[B \mid A] + \mathbf{P}[C \mid A]) = Geom(1)$
5. $\mathbb{E}(M_A) = 1$
6. *Similarly, we have* $\mathbb{E}(M_B) = \frac{5}{4}$, $\mathbb{E}(M_A) = \frac{9}{4}$

Thus, for the Basic PNS Chain $\{J_k, M_k\}_{k=1}^K$ with large K, the proportions $\mathcal{P}$ of state A, B, and C in the Markov chain are

$$\begin{aligned} \mathcal{P}_{Basic\ PNS}(A) &= \frac{\sum_{J_k=A} M_k}{\sum_{k=1}^K M_k} = \frac{1}{1 + \frac{5}{4} + \frac{9}{4}} = \frac{2}{9} \neq \pi(A) = \frac{1}{6}; \\ \mathcal{P}_{Basic\ PNS}(B) &= \frac{5}{18} \neq \pi(B) = \frac{1}{3}; \\ \mathcal{P}_{Basic\ PNS}(C) &= \frac{1}{2} = \pi(C). \end{aligned} \tag{4.2}$$

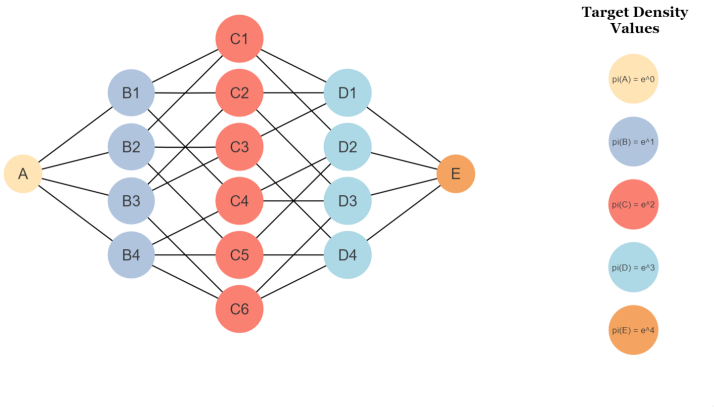

Figure 4.2: Diagram of Example 2 showing non-convergence property of the Basic PNS.

This example shows that the samples from Basic PNS are not converging to the target density π.

Example 12. *The other example illustrating the non-convergence property of Basic PNS is shown in Figure 4.2, which is a much larger problem compared to the first example. We have 16 states in example 2. All States are connected to exactly four states. The target density is described as* $\pi(A) \propto 1$, $\pi(B_1) = \pi(B_2) = \pi(B_3) = \pi(B_4) \propto e$, $\pi(C_1) = \pi(C_2) = \cdots = \pi(C_6) \propto e^2$, $\pi(D_1) = \pi(D_2) = \pi(D_3) = \pi(D_4) \propto e^3$, *and* $\pi(E) \propto e^4$. *This example is too large to be calculated by hand, so we use simulations to calculate the limiting distribution of the samples. The convergence of the sampling distribution is measured by the Total Variation Distance (TVD) again, where*

$$TVD(\mathcal{P}_{Sampled}, \pi) = \frac{1}{2} \sum_{x \in \mathcal{S}} \left| \mathcal{P}_{Sampled}(x) - \pi(x) \right|. \tag{4.3}$$

Convergence to stationarity is described by how quickly TVD decreases to 0.

The simulation results are shown in Figure 4.3. For a given amount of samples ($K = 50$, 100, 150, 200, ..., 500, 1000, 1500, 2000, ..., 7500), we did 1000 simulations for each of them. The TVD values and the times here are the average values from these 1000 simulations. We compared four methods: Rejection Free and Basic PNS with three different subset sizes. The Markov chains, produced by Rejection-Free, will converge to the target density, so the TVD value gets close to 0 at last. For PNS with $|\mathcal{N}_0| = 1$, we select one random neighbor among all four neighbors at a time, forcing the chain to move to that state. This method is the worst, and it converges at around 0.3. PNS with $|\mathcal{N}_0| = 2$ means that we randomly select two neighbors at every step and apply the Rejection-Free technique (select from these two states by probability proportional to the transition probability, and calculate the multiplicity list by the average of the transition probabilities). All three PNS algorithms are not converging to the target density π.

Both Examples show that the samples from Basic PNS will not converge to the target distribution π. Thus, we turn attention to a more promising avenue, the unbiased version of the Partial Neighbor Search algorithm, where convergence to stationarity is guaranteed.

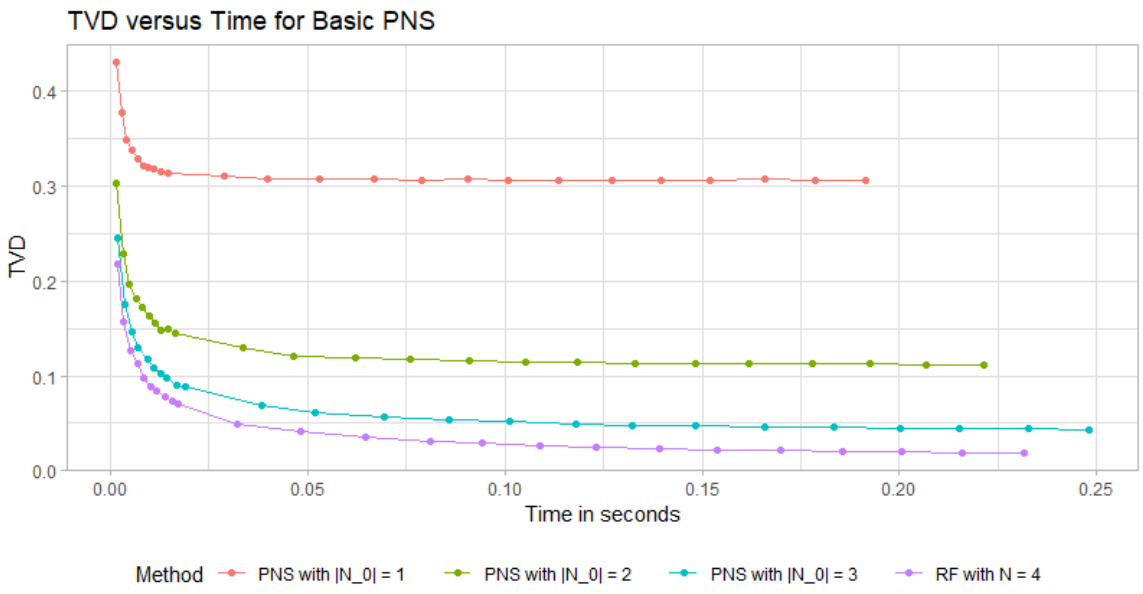

Figure 4.3: Average values of TVD between samples and the target density π for Example 2 as a function of average CPU time in seconds for four scenarios: Rejection-Free and Basic PNS with three different Partial Neighbor Set sizes. Each dot within the plot represents the result of the average TVD value and average CPU time in seconds from 1000 simulation runs given a certain original sample size, where the sizes are $\{50, 100, 150, 200, \ldots, 500, 1000, 1500, 2000, \ldots, 7500\}$.

4.2 Unbiased Partial Neighbor Search algorithm

4.2.1 Hybrid Chains for Partial Neighbor Search

We talked about the hybrid chains technique for the Metropolis-Hastings algorithm in Section 2.3.1 and the Rejection-Free algorithm in Section 3.5. The technique from the hybrid chains can also help us to make PNS chains converge to the target density correctly. We first define the meaning of Partial Neighbor Sets here. For simplicity, we focus on discrete cases here and will define the Partial Neighbor Sets for general state space in Section 4.5.

Before we start our Markov chain, we have a proposal distribution $\mathcal{Q}$ with a corresponding neighbor set $\mathcal{N}$ where $\mathcal{N}(x) := \{y \in \mathcal{S} \mid \mathcal{Q}(x, y) > 0\}$. A Partial Neighbor Set means any function $\mathcal{N}_i$ satisfies the following conditions:

1. $\mathcal{N}_i : \mathcal{S} \to \mathbb{P}(\mathcal{S})$, where $\mathcal{S}$ is the state space, and $\mathbb{P}(\mathcal{S})$ is the power set of $\mathcal{S}$;
2. $\mathcal{N}_i(x) \subset \mathcal{N}(x)$, $\forall x \in \mathcal{S}$;
3. $y \in \mathcal{N}_i(x) \iff x \in \mathcal{N}_i(y)$, $\forall x, y \in \mathcal{S}$;

Usually, we want to pick $\mathcal{N}_i$ such that $|\mathcal{N}_i(x)| < |\mathcal{N}(x)|$ to perform proper PNS. In addition, to ensure irreducibility, we need to make sure $\cup_{i=0}^{\mathcal{I}-1} \mathcal{N}_i(x) = \mathcal{N}(x)$ for all $x \in \mathcal{S}$. Note that, $\cup_{i=0}^{\mathcal{I}-1} \mathcal{N}_i(x) = \mathcal{N}(x)$ for all $x \in \mathcal{S}$ is sufficient to prove that the irreducibility, but not necessary. The corresponding proposal distribution is defined to be $\mathcal{Q}_i(x, y) : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, where $\mathcal{Q}_i(x, y) \propto \mathcal{Q}(x, y)$ for $y \in \mathcal{N}_i(x)$ and $\mathcal{Q}_i(x, y) = 0$ otherwise. Note that, we used superscripts in Section 3.5 for the proposal

distribution Q for Rejection-Free, and we use subscripts for the corresponding proposal distribution Q for Partial Neighbor Search. We used different notations for the proposal distributions to show their difference. For Rejection-Free, the Markov chain produced by each proposal distribution Q^i with superscripts will converge to the target distribution with just themselves, and we want to use the combination of them to improve the efficiency of convergence. On the other hand, the proposal distribution Q_i with subscripts for Partial Neighbor Search will not converge to the target distribution with only themselves, and we need to combine all of them to make the samples converge to the target distribution. Both of these two notations Q^i and Q_i use i as the index for different proposal distributions. The same thing happens to the corresponding neighbor sets $\mathcal{N}_i$

Therefore, we propose the Unbiased Partial Neighbor Search (Unbiased PNS) with Hybrid Chains for every L_0 original samples as shown in Algorithm 7. The proof that the Markov chain produced by Unbiased PNS will converge to the target distribution π is shown in Section 3.8. Note that, the partial neighbor set $\mathcal{N}_i(J_{k-1})$ used at each step are defined by two things, the current state $J_{k-1} \in \mathcal{S}$ and the partial neighbor set function $\mathcal{N}_i : \mathcal{S} \to \mathbb{P}(\mathcal{S})$, where $\mathcal{S}$ is the state space, and $\mathbb{P}(\mathcal{S})$ is the power set of $\mathcal{S}$. At each step, our current state J_{k-1} will be different. In addition, after we obtain a certain number of original samples, we will need to switch to a new partial neighbor set function $\mathcal{N}_i$ for $i = 0, 1, \ldots, \mathcal{I} - 1$, where all functions $\mathcal{N}_i$ are determined at the very beginning.

Algorithm 7 Unbiased Partial Neighbor Search

select $\mathcal{N}_i$ for $i = 0, 1, \ldots, \mathcal{I} - 1$ where $\cup_{i=0}^{\mathcal{I}-1} \mathcal{N}_i(X) = \mathcal{N}(X)$
initialize $i \leftarrow 0$ ▷ start with proposal distribution $\mathcal{Q}_0$
initialize $L \leftarrow L_0$ ▷ start with L_0 remaining original samples
initialize J_0
for k in 1 to K **do**
calculate multiplicity list $m \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}_i(J_{k-1})} \mathcal{Q}_i(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}_i(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}_i(J_{k-1}, z)}\right\}$$

if $m \leq L$ **then** ▷ if we have enough remaining original samples
$M_{k-1} \leftarrow m$, $L \leftarrow L - m$
choose the next jump chain State $J_k \in \mathcal{N}_i(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}_i(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}_i(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}_i(y, J_{k-1})}\right\}$$

else ▷ if we don't have enough remaining original samples
$M_{k-1} \leftarrow L$, $L \leftarrow L_0$, $J_k \leftarrow J_{k-1}$, $i \leftarrow (i + 1 \mod \mathcal{I})$
▷ stay at J_{k-1} for L times and switch to the next $\mathcal{N}_i$
end if
end for

Again, in Algorithm 7, when we need to pick our next state according to the given probabilities, we can use the technique shown in Section 3.8, which is faster than other methods to sample proportionally. The Markov chains produced by Algorithm 7 will converge to the target distribution, but how is its efficiency compared to the Metropolis-Hasting algorithm and Rejection-Free? We will compare these three algorithms in Section 4.3.

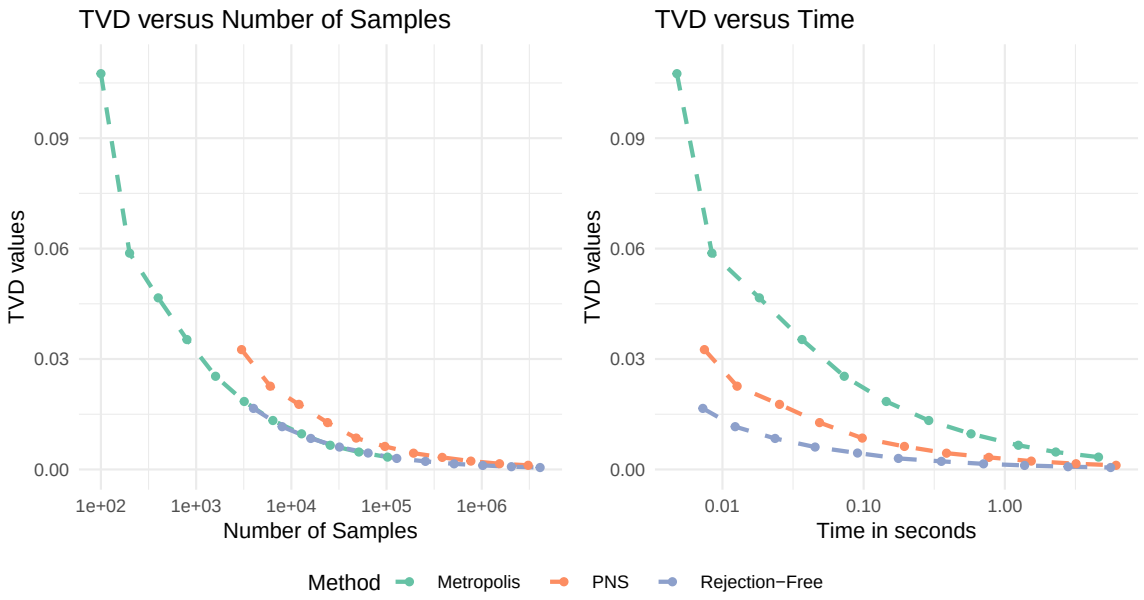

Figure 4.4: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for three methods: Metropolis algorithm, Rejection-Free, and Unbiased PNS. We used an upper triangular 16×16 QUBO matrix, generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the result of the average TVD value and time used for 1000 simulation runs given certain original sample sizes. The original sample sizes for the Metropolis algorithm are $\{100, 200, 400, 800, \ldots, 1024000\}$. The original samples from Rejection-Free are 40x more than those from Metropolis, and the original samples from Unbiased PNS are 30x more than those from Metropolis. We choose these sizes to get a close average CPU time for all three methods. For Unbiased PNS, we used $|\mathcal{N}_k| = 8$ and $L_0 = 100$.

4.3 Application to QUBO model

Quadratic unconstrained binary optimization (QUBO) has been rising in importance in combinatorial optimization because of its wide range of applications in finance and economics to machine learning [48]. It can also be used as a sampling question, which aims to sample from the distribution

$$\pi(x) = \exp\{x^T Q x\}, \text{ where } x \in \{0,1\}^N \tag{4.4}$$

for a given N by N matrix Q (usually symmetric or upper triangular).

To run our algorithm, we used uniform proposal distributions among all neighbors where the neighbors are defined as binary vectors with Hamming distance 1. That is, $\mathcal{Q}(x, y) = \frac{1}{N}$ for $\forall y$ such that $|x - y| = \sum_{i=1}^{N} |x_i - y_i| = 1$, $\forall x, y \in \{0,1\}^N$. Thus, the neighbors are all binary vectors different by one flip. For the first simulation here, the PNS neighbor sets $\mathcal{N}_0$, $\mathcal{N}_1$ are chosen systematically, where $\mathcal{N}_0$ represents flip entries from 1 to $\lfloor \frac{N}{2} \rfloor$, and $\mathcal{N}_1$ represents flip entries from $\lfloor \frac{N}{2} \rfloor + 1$ to N. We will discuss many other choices for the PNS neighbor sets in Section 4.6

Figure 4.4 shows the results for comparing the Metropolis algorithm, Rejection-Free, and Unbiased PNS by sampling from a 16×16 QUBO question to a single-core implementation. The QUBO matrix Q is an upper triangular matrix, where the non-zero elements were generated randomly

by $Q_{i,j} \sim \text{Normal}(0, 10^2)$, $\forall i \leq j$. We compare the TVD values from the Metropolis Algorithm, Rejection-Free, and Unbiased PNS with different original sample sizes. For the Metropolis algorithm, the numbers of original samples are $\{100, 200, 400, 800, \ldots, 102400\}$. The number of original samples for Rejection-Free is 40 times more than the number for the Metropolis algorithm, and the number for Unbiased PNS is 30 times more. We used these many numbers of original samples to make the run-time for all three algorithms to be about the same. For each given number of original sample sizes, we simulated 1000 runs, recorded the corresponding TVD values and times used for the sampling part, and calculated the average values given the number of original samples. Note that the average time represents the CPU time for where the algorithm is calculated by running the algorithm on a single-core implementation. In addition, before we generate the samples, we apply the algorithm for the same number of steps for burn-in. For example, when we create a Markov chain with 102400 original samples, we first generate 102400 original samples for burn-in.

From Figure 4.4, we can see that the quality of the samples by the Metropolis algorithm and Rejection-Free are the same given the original sample sizes. This result is consistent with our conclusion that Rejection-Free is identical to the Metropolis algorithm, except Rejection-Free generates the same states simultaneously with all immediately repeated states. Thus, these two algorithms are different only by the CPU time. In addition, the quality of the samples by Unbiased PNS is worse than both the Metropolis algorithm and Rejection-Free given a certain number of original samples because each Partial Neighbor Set is biased within its L_0 original samples, while the combination of them is unbiased. Thus, the average TVD value from Unbiased PNS is more larger given the same amount of original samples. However, for a given amount of CPU time, the performance of Unbiased PNS is much better than the Metropolis algorithm and worse than Rejection-Free.

In this case, Unbiased PNS can provide significant speedups compared to the Metropolis algorithm. On the other hand, we did not expect the Unbiased PNS can beat Rejection-Free under this circumstance. Unbiased PNS is worse than Rejection-Free in two aspects. First, the Unbiased PNS is biased within each L_0 original samples. In addition, at the end of each L_0 original samples, the algorithm is very likely to reject once and stay in the same state. Thus, Unbiased PNS is not entirely rejection-free anymore and usually rejects once for every L_0 original samples.

However, we need the Unbiased PNS because we may not have as many circuit blocks in the parallel hardware as we want. Thus, we can, at most, consider a limited number of neighbors for some specialized hardware, such as DA. Thus, Rejection-Free is not applicable in this case, and we would need the help of Unbiased PNS, which is better than applying the Metropolis algorithm.

Again, parallel hardware can increase the speed for both Rejection-Free and Unbiased PNS by mapping the calculation of the transition probabilities for different neighbors onto different cores. Besides that, we can also use multiple replicas at different temperatures, such as in parallel tempering, or deploy a population of replicas at the same temperature [76]. Combining these methods by parallel can yield 100x to 10,000x speedups for both Rejection-Free and Unbiased PNS [76].

4.4 Continuous Models and the Donuts Example

We talked about the application of Unbiased PNS to discrete cases in the previous sections. Can we apply Unbiased PNS on continuous models? We first review how to apply Rejection-Free on general (continuous) state space as in Theorem 3.3.6.

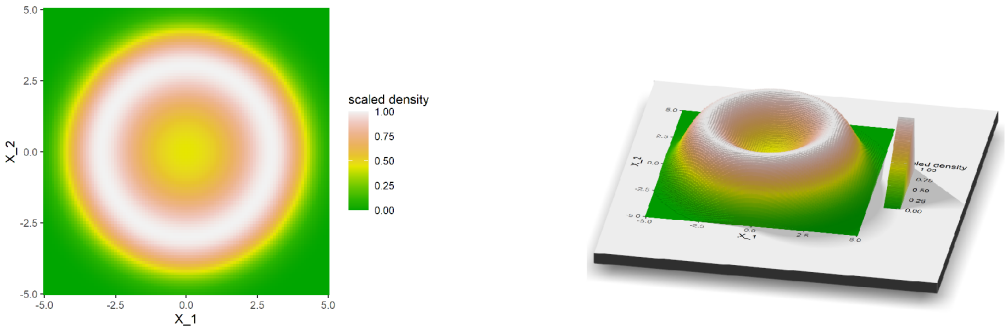

Figure 4.5: The scaled probability density plot for the Donuts Example with $\mu_0 = 9$ and $\sigma = 10$. The density is scaled to $[0, 1]$. We used large σ to show the shape of our distribution. With small σ, it is hard to see the shape of a sharply peaked distribution.

Although we have a solid theory base for Rejection-Free on general state space, applying Rejection-Free to the continuous sampling questions efficiently on most computer hardware is pretty hard. The biggest challenge is the calculation of integration $\int q(x, z)\mu(dz)$. We need many calculations for the numerical integration. In addition, such tasks can hardly be split efficiently into specialized hardware with a reasonable amount of parallel calculating units. At the same time, Unbiased PNS can be surprisingly helpful in this case. As long as the Metropolis algorithm can be applied, PNS can be applied straightforwardly without any calculation of integration. We need to choose the Partial Neighbors Sets $\mathcal{N}_i(x)$ to be a finite subset of all the neighbors $\mathcal{N}(x)$ in Algorithm 7. We check the performance of our Unbiased PNS on a simple continuous sampling question: the Donuts Example.

Inspired by [26], we use a donuts example to show Unbiased PNS's performance on continuous state space. Suppose we have two independent random variables μ and θ where

$$\mu \sim \text{Normal}^+(\mu_0, \sigma^2), \ \theta \sim \text{Uniform}[0, \pi). \tag{4.5}$$

Here, Normal$^+$ means the Truncated Normal distribution without the negative tail, and π in the Uniform distribution means the circular constant instead of the target density. Then we define two random variables X_1 and X_2 to be

$$X_1 = \sqrt{\mu}\sin\theta, \ X_2 = \sqrt{\mu}\cos\theta. \tag{4.6}$$

The determinant of the Jacobian matrix is $\frac{1}{2}$. Thus we have

$$f_{X_1,X_2}(x_1, x_2) \propto \frac{1}{\sigma}\exp\left[-\frac{(x_1^2 + x_2^2 - \mu_0)^2}{2\sigma^2}\right], \tag{4.7}$$

For example, a 3-D map for the density for X_1 and X_2 with $\mu_0 = 9$, and $\sigma = 10$ is shown in Figure 4.5. The density is scaled to $[0, 1]$. In our later simulation, we use $\mu_0 = 9$ and $\sigma = 0.1$ instead. We use large σ to show the shape of our distribution because it is hard to see its shape when it sharply peaks with a small σ. However, for the simulation, PNS can outperform the Metropolis algorithm

when there are many rejections, so we use a small σ to get a sharply peaked distribution to increase the rejection rate in the Metropolis algorithm. Note that Unbiased PNS and Rejection-Free are not always better than the Metropolis algorithm. For an extreme example, when we have a distribution where all the states have the same target density values, there will be no rejection for the Metropolis algorithm. At each step, the Metropolis algorithm will uniformly pick a random neighbor from the current state and move to that neighbor, while (Rejection-Free / PNS) will calculate the transition probabilities for (all / part) of the neighbors and uniformly pick a random one. The Metropolis algorithm will be far better than Rejection-Free and PNS in this case. In practice, the higher the dimension of the problem and the more sharply peaked the distribution is, the better the Rejection-Free and Unbiased PNS will be. Thus, we use $\sigma = 0.1$ for the simulation to create a sharply peaked distribution for later simulation.

In addition, the proposal distribution is defined to be the standard normal distribution for both dimensions. That is, for $x = (x_1, x_2), y = (y_1, y_2) \in \mathbb{R}^2$, $\mathcal{Q}(x, y) = \phi(y_1 - x_1)\phi(y_2 - x_2)$ where ϕ is the density function of the standard normal distribution. Then for any $x \in \mathbb{R}^2$, we have $\mathcal{N}(x) = \mathbb{R}^2$. For PNS, L_0 is selected to be 1000. Using $L_0 = 100$ to 1000 will not affect the sampling speed too much, similar to the conclusion in Section 4.6.

Since the Partial Neighbor Sets are always the whole space of $\mathbb{R}^2$, it is tough for us to apply Rejection-Free here since the integration of the whole space needs too many computational resources. Even if we limit the neighbors to a small area around the current state, integration is needed as long as the problem is continuous, and the Rejection-Free will be consequentially slow. At the same time, Unbiased PNS can be applied to continuous cases without calculating integration by making minor changes to Algorithm 7. The Unbiased PNS algorithm for continuous is stated as Algorithm 8.

In Algorithm 8, we did not define the systematic Partial Neighbor Sets as we had for the discrete cases. We want to use Unbiased PNS with finite many partial neighbors being considered at each step, but we have uncountable neighbors. It is impossible to divide these uncountable neighbors into finite partial neighbor sets with finite sizes. Thus, unlike for the discrete case, we divide the whole neighbor set into $\mathcal{I}$ Partial Neighbor Sets, for the continuous case, we use the random Partial Neighbor Set, which randomizes a new finite partial neighbor set for every L_0 original samples. The notation here is then different as well. For discrete cases, we divide the whole neighbor set $\mathcal{N}$ into $\mathcal{N}_i$ for $i \in \{0, 1, 2, \ldots, \mathcal{I} - 1\}$ with $\cup_{i=0}^{\mathcal{I}-1} \mathcal{N}_i = \mathcal{N}$. On the other hand, we use one single notation $\mathcal{N}_0$ to represent our Partial Neighbor Set in continuous cases, and it will be redrawn for every L_0 original samples. In the later simulation, we use partial neighbor sets with $|\mathcal{N}_0| = 50$. That is, we consider 50 partial neighbors at each step. Note that, in Section 4.2.1, we defined the Partial Neighbor Sets, and according to the third condition, we must have reversibility for $\mathcal{N}_0(x)$, which means $y \in \mathcal{N}_0(x) \iff x \in \mathcal{N}_0(y), \forall x, y \in \mathcal{S}$. Therefore, we choose the Partial neighbor Set $\mathcal{N}_0(x)$ as follows:

1. generate $\delta_1, \delta_2 \sim \text{Normal}(0, 1)$;
2. for state $x = (x_1, x_2)$, put $y = (x_1 + \delta_1, x_2 + \delta_2)$ into the Partial Neighbor Set $\mathcal{N}_i(x)$;
3. to ensure the reversibility, also put $y' = (x_1 - \delta_1, x_2 - \delta_2)$ into the Partial Neighbor Set $\mathcal{N}_i(x)$;
4. repeats the above steps 25 times to generate a total of 50 neighbors for the Partial Neighbor Set $\mathcal{N}_0(x)$.

Algorithm 8 Unbiased PNS for Continuous Case

select one Partial Neighbor Set $\mathcal{N}_0$
initialize $L \leftarrow L_0$ ▷ start with L_0 remaining original samples
initialize J_0
for k in 1 to K **do**
calculate multiplicity list $m \leftarrow 1 + G$ where $G \sim \text{Geometric}(p)$ with

$$p = \sum_{z \in \mathcal{N}_0(J_{k-1})} \mathcal{Q}_0(J_{k-1}, z) \min\left\{1, \frac{\pi(z)\mathcal{Q}_0(z, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}_0(J_{k-1}, z)}\right\}$$

if $m \leq L$ **then** ▷ if we have enough remaining original samples
$M_{k-1} \leftarrow m$, $L \leftarrow L - m$
choose the next jump chain State $J_k \in \mathcal{N}_0(J_{k-1})$ such that

$$\hat{P}(J_k = y \mid J_{k-1}) \propto \mathcal{Q}_0(J_{k-1}, y) \min\left\{1, \frac{\pi(y)\mathcal{Q}_0(y, J_{k-1})}{\pi(J_{k-1})\mathcal{Q}_0(y, J_{k-1})}\right\}$$

else ▷ if we don't have enough remaining original samples
$M_{k-1} \leftarrow L$, $L \leftarrow L_0$, $J_k \leftarrow J_{k-1}$,
▷ stay at J_{k-1} for the remaining L times
select a new Partial Neighbor Set $\mathcal{N}_0$
end if
end for

Moreover, we measure the sampling results by bias instead of the TVD. The calculation of TVD in the continuous case also needs much integration, which is hard to calculate. On the other hand, given samples $\{X_1, X_2 \ldots, X_K\}$, we usually use the MCMC to approximate the expected value $\mathbb{E}_\pi(h)$ of a function $h : S \to \mathbb{R}$ by the usual estimator, $\hat{e}_K(h) = \frac{1}{K}\sum_{k=1}^{K} h(X_{1,k}, X_{2,k})$. The Strong Law of Large Numbers for Markov chains says that assuming that $\mathbb{E}_\pi(h)$ is finite and that the Markov chain is irreducible with stationary distribution π, we must have $\lim_{K\to\infty} \hat{e}_K = \mathbb{E}_\pi(h)$. Therefore, $\text{Bias}(h) = |\hat{e}_K(h) - \mathbb{E}_\pi(h)| = |\frac{1}{K}\sum_{k=1}^{K} h(X_k) - \mathbb{E}_\pi(h)|$ can also be a good measurement for the quality of the samples. According to the definition, bias is greater or equal to 0. When the samples $\{X_1, X_2 \ldots, X_K\}$ gets closer to the target distribution π, the bias will decrease to 0. Thus, convergence to stationarity is described by how quickly the bias decreases to 0 for all function h. This property is similar to TVD from Section 4.3. In fact, for any probability distribution $\mathcal{P}_1$ and $\mathcal{P}_2$, $\text{TVD}(\mathcal{P}_1, \mathcal{P}_2) = \sup_{\mathcal{S}} \left(\mathcal{P}_1(\mathcal{S}), \mathcal{P}_2(\mathcal{S})\right)$ [13].

For example, we check the sum of the bias from the first-degree terms X_1 and X_2. Since the Donuts example is centered at 0, thus $\mathbb{E}_\pi(X_1) = \mathbb{E}_\pi(X_2) = 0$. Thus, we have

$$\begin{aligned} \text{Bias}(X_1) + \text{Bias}(X_2) &= |\hat{e}_K(X_1) - \mathbb{E}_\pi(X_1)| + |\hat{e}_K(X_2) - \mathbb{E}_\pi(X_2)| \\ &= \left|\frac{1}{K}\sum_{k=1}^{K} X_{1,k}\right| + \left|\frac{1}{K}\sum_{k=1}^{K} X_{2,k}\right|, \end{aligned} \tag{4.8}$$

Note that both biases will decrease to 0 since our Markov chain converges to the target density π. In addition, for the Rejection-Free Chain $\{J_{1,k}, J_{2,k}, M_k\}_{k=1}^{K}$ generated by the Unbiased PNS

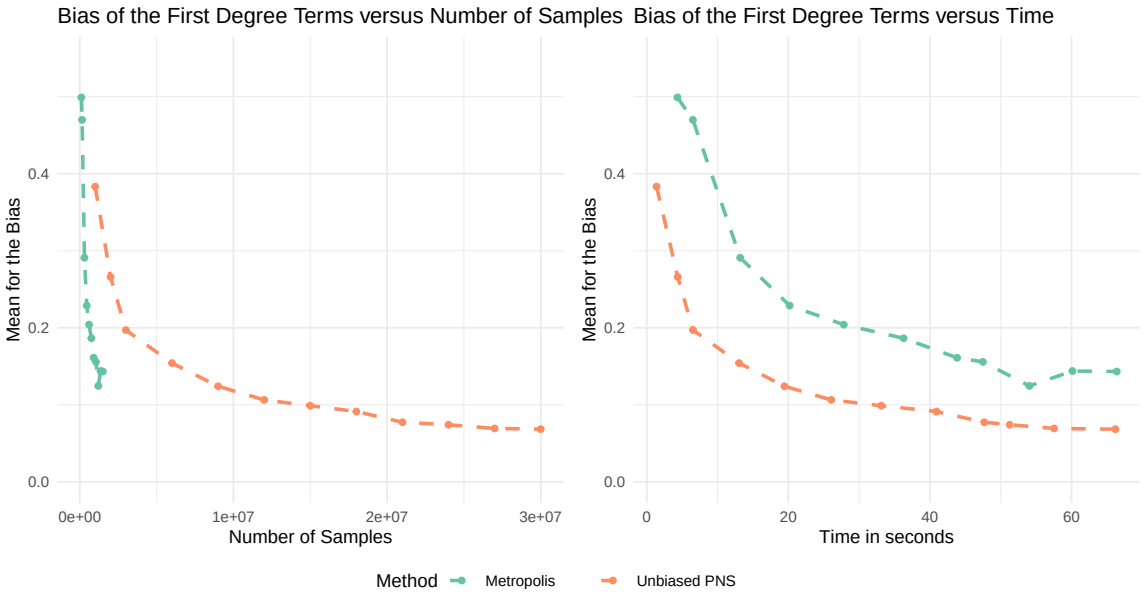

Figure 4.6: Sum of the Average Bias of X_1 and X_2 between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for two methods: Metropolis algorithm and Unbiased PNS. We used the Donuts example with $\mu_0 = 9$ and $\sigma = 0.1$. Each dot within the plot represents the result of the average bias value and time used for 1000 simulation runs given certain original sample sizes. The original sample sizes for the Metropolis algorithm are $\{50000, 100000, 150000, 300000, 450000, \ldots, 1500000\}$. The sizes for Unbiased PNS are 20x more than the sizes for the Metropolis. We choose these sizes to get a close average CPU time for both methods. For Unbiased PNS, we used $|\mathcal{N}_k| = 50$ and $L_0 = 1000$.

algorithm, the bias is defined to be

$$\text{Bias}(J_1) + \text{Bias}(J_2) = \frac{\left|\sum_{k=1}^{K} M_k \times J_{1,k}\right|}{\sum_{k=1}^{K} M_k} + \frac{\left|\sum_{k=1}^{K} M_k \times J_{2,k}\right|}{\sum_{k=1}^{K} M_k} \tag{4.9}$$

The result for comparing the Metropolis algorithm and Unbiased PNS by the bias of first-degree term is shown in Figure 4.6. Each dot within the plot represents the average value of 100 simulation runs. For each run, we generate a Markov chain for a given number of samples for both algorithms. The average time represents the CPU time we apply the algorithm by a single-core implementation. Again, parallel hardware such as DA can yield 100x to 10,000x speedups for Unbiased PNS [76].

From Figure 4.6, we can see that the quality of the samples by Unbiased PNS is again worse than the Metropolis algorithm because each Partial Neighbors Set is biased within L_0 original samples, while the combination of them is unbiased. Thus, the average bias values for Unbiased PNS are more significant for the same amount of samples. However, for a given amount of CPU time, the performance of Unbiased PNS is much better than the Metropolis algorithm. For this example, the Unbiased PNS can get 30x more samples than the Metropolis algorithm within the same time by a single-core implementation. Rejections slow down the Metropolis algorithm while Unbiased PNS is not influenced, and thus, Unbiased PNS works much better in this simulation.

In addition, we can also check the sum of the bias from the second-degree terms $\text{Bias}(X_1^2) +$

$\text{Bias}(X_2^2)$, the sum of the bias from the fourth-degree terms $\text{Bias}(X_1^4)+\text{Bias}(X_2^4)$, and the sum of the bias from the positive rate $\text{Bias}(\mathbb{1}(X_1>0))+\text{Bias}(\mathbb{1}(X_1>0))$, where $\mathbb{1}$ means the indicator function. To calculate the bias of the second-degree terms, we have $X_1^2+X_2^2=\mu\sim\text{Normal}^+(\mu_0,\sigma^2)$. Note that, for the Truncated normal distribution with mean 9 and standard deviation 0.1, the probability for a negative tail is too small, so we can treat it as a normal distribution. Thus,

$$\mathbb{E}_\pi(X_1^2)=\frac{1}{2}\mathbb{E}_\pi(X_1^2+X_2^2)=\frac{1}{2}\mathbb{E}_\pi(\mu^2)\approx\frac{1}{2}\mu_0^2. \tag{4.10}$$

$$\begin{aligned}\text{Bias}(X_1^2)+\text{Bias}(X_2^2)&=|\hat{e}_K(X_1^2)-\mathbb{E}_\pi(X_1^2)|+|\hat{e}_K(X_2^2)-\mathbb{E}_\pi(X_2^2)|\\&\approx|\frac{1}{K}\sum_{k=1}^K X_{1,k}^2-\frac{1}{2}\mu_0^2|+|\frac{1}{K}\sum_{k=1}^K X_{2,k}^2-\frac{1}{2}\mu_0^2|.\end{aligned} \tag{4.11}$$

Similarly,

$$\begin{aligned}\text{Bias}(X_1^4)+\text{Bias}(X_2^4)=&|\hat{e}_K(X_1^4)-\mathbb{E}_\pi(X_1^4)|+|\hat{e}_K(X_2^4)-\mathbb{E}_\pi(X_2^4)|\\\approx&|\frac{1}{K}\sum_{k=1}^K X_{1,k}^4-\frac{3}{8}(\mu_0^4+\sigma^2)|+\\&|\frac{1}{K}\sum_{k=1}^K X_{2,k}^4-\frac{3}{8}(\mu_0^4+\sigma^2)|;\end{aligned} \tag{4.12}$$

$$\begin{aligned}\text{Bias}(\mathbb{1}(X_1>0))+\text{Bias}(\mathbb{1}(X_2>0))=&|\hat{e}_K(\mathbb{1}(X_1>0))-\mathbb{E}_\pi(\mathbb{1}(X_1>0))|+\\&|\hat{e}_K(\mathbb{1}(X_2>0))-\mathbb{E}_\pi(\mathbb{1}(X_2>0))|\\=&|\frac{1}{K}\sum_{k=1}^K\mathbb{1}(X_{1,k}>0)-\frac{1}{2}|+\\&|\frac{1}{K}\sum_{k=1}^K\mathbb{1}(X_{2,k}>0)-\frac{1}{2}|.\end{aligned} \tag{4.13}$$

The results for the comparison of the Metropolis algorithm and Unbiased PNS by the sum of the average bias from the second-degree terms $\text{Bias}(X_1^2)+\text{Bias}(X_2^2)$, the fourth-degree terms $\text{Bias}(X_1^4)+\text{Bias}(X_2^4)$, and the positive rate $\text{Bias}(\mathbb{I}(X_1>0))+\text{Bias}(\mathbb{I}(X_1>0))$ are shown in Figure 4.7. From the result for different choices of the terms, we can conclude that Unbiased PNS performs better than the Metropolis algorithm in this continuous Donuts example.

4.5 Convergence Theorem

Now, we will prove the convergence of the PNS algorithm in the general state space case.

Definition 4.5.1. *For sampling questions in general state space, we usually have the following elements:*

1. *a state space $\mathcal{S}$;*
2. *a σ-finite reference measure μ on $\mathcal{S}$, where μ could be a counting measure for discrete cases, and μ could be a Lebesgue measure for continuous cases;*
3. *a target density $\pi:\mathcal{S}\to[0,1]$, where $\int_{x\in\mathcal{S}}\pi(x)\mu(dx)=1$;*

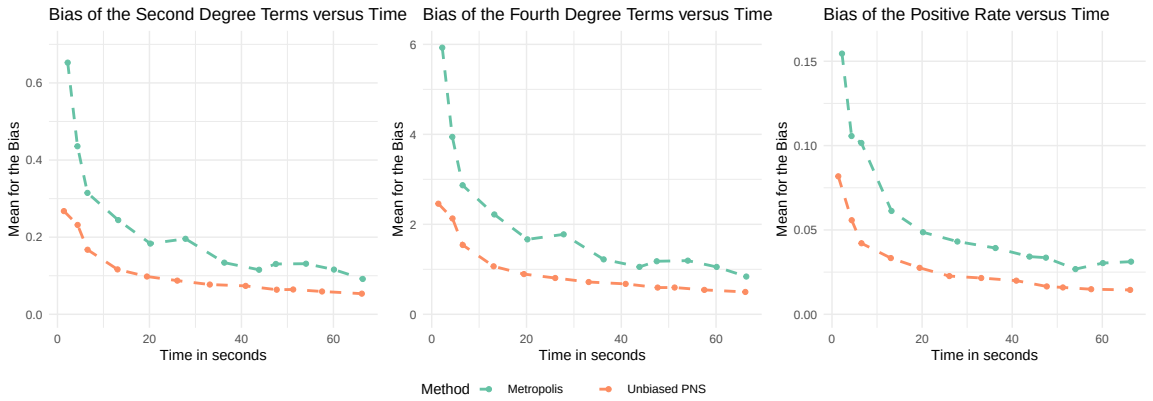

Figure 4.7: Sum of the average bias from the second-degree terms $\text{Bias}(X_1^2) + \text{Bias}(X_2^2)$ (left), the fourth-degree terms $\text{Bias}(X_1^4) + \text{Bias}(X_2^4)$ (middle), and the positive rate $\text{Bias}(\mathbb{1}(X_1 > 0)) + \text{Bias}(\mathbb{1}(X_1 > 0))$ (right) between sampling and target density π as a function of average time in seconds for two methods: Metropolis algorithm and Unbiased PNS. $\mathbb{I}$ means the indicator function. We used the Donuts example with $\mu_0 = 9$ and $\sigma = 0.1$. Each dot within the plot represents the result of the average bias value and time used for 1000 simulation runs given certain original sample sizes. The original sample sizes for the Metropolis algorithm are $\{50000, 100000, 150000, 300000, 450000, \ldots, 1500000\}$. The sizes for Unbiased PNS are 20x more than the sizes for the Metropolis. We choose these sizes to get a close average CPU time for both methods. For Unbiased PNS, we used $|\mathcal{N}_k| = 50$ and $L_0 = 1000$.

4. *a target distribution* $\pi : \mathbb{P}(\mathcal{S}) \to [0, 1]$, *where* $\pi(\mathcal{A}) \coloneqq \int_{\mathcal{A}} \pi(x)\mu(dx)$, $\forall \mathcal{A} \subset \mathcal{S}$, *and* $\mathbb{P}$ *means the power set;*

5. *a proposal density* $q(x, y) : \mathcal{S} \times \mathcal{S} \to [0, 1]$, *where* $\int_{\mathcal{S}} q(x, y)\mu(dy) = 1$, $\forall x, y \in \mathcal{S}$;

6. *a proposal distribution* $\mathcal{Q}(x, dy) \propto q(x, y)\mu(dy)$;

7. *a corresponding neighbor set* $\mathcal{N}(x) \coloneqq \{y \in \mathcal{S} \mid q(x, y) > 0\} \subset \mathcal{S} \backslash \{x\}$;

8. *the transition probabilities* $\mathbf{P}(x, dy) = q(x, y) \min\left(1, \frac{\pi(y)q(y,x)}{\pi(x)q(x,y)}\right)\mu(dy)$; $\mathbf{P}(x, dy) = q(x, y)\mu(dy)$ *if the denominator* $\pi(x)q(x, y) = 0$.

Given the above elements, assume irreducibility and aperiodicity, we can generate a Markov chain $\{X_0, X_1, \ldots, X_K\}$ *such that the limiting distribution of* $\lim_{n\to\infty} X_n$ *converges to the stationarity distribution* $\pi(x)\mu(dx)$ *by Algorithm 1.*

Definition 4.5.2. *Suppose we have a state space* $\mathcal{S}$, *a reference measure* μ, *and a target density* π, *the proposal distribution* $\mathcal{Q}$ *and the corresponding neighbor set* $\mathcal{N}$. *Then, a Partial Neighbor Set* $\mathcal{N}_i$ *means a function* $\mathcal{N}_i$ *satisfying the following conditions:*

1. $\mathcal{N}_i : \mathcal{S} \to \mathbb{P}(\mathcal{S})$, *where* $\mathcal{S}$ *is the state space, and* $\mathbb{P}(\mathcal{S})$ *is the power set of* $\mathcal{S}$;

2. $\mathcal{N}_i(x) \subset \mathcal{N}(x)$, $\forall x \in \mathcal{S}$, *and we must pick a finite subset* $\mathcal{N}_i(x)$ *to ensure a finite for loop in Algorithm 7;*

3. $y \in \mathcal{N}_i(x) \iff x \in \mathcal{N}_i(y)$, $\forall x, y \in \mathcal{S}$;

Given a Partial Neighbor Set $\mathcal{N}_i$, the proposal distribution for $\mathcal{N}_i$ is defined to be $\mathcal{Q}_i : \mathcal{S} \times \mathbb{P}(\mathcal{S}) \to \mathbb{R}$, where $\mathcal{Q}_i(x, dy) = \frac{\sum_{r \in \mathcal{N}_i(x)} q(x,r)\delta_r(dy)}{\sum_{z \in \mathcal{N}_i(x)} q(x,z)}$, where δ_r means the point mass at r. On the other hand, since we will only pick a finite subset $\mathcal{N}_i(x)$, $\mathcal{Q}_i(x, dy)$ can also be expressed as $\mathcal{Q}_i(x, y) = \frac{q(x,y)\mathbb{1}(y \in \mathcal{N}_i(x))}{\sum_{z \in \mathcal{N}_i(x)} q(x,z)}$, where $\mathbb{1}$ means indicator function.

Here, before we prove the convergence theorem of the Unbiased PNS as stated in Algorithm 7, we first prove it for another version of the Unbiased PNS as stated in Algorithm 9. It is easy to see that the only difference between Algorithm 9 and Algorithm 7 is that we are not using the Rejection-Free technique here, where we calculate all the transition probabilities at once, pick the next jump chain state, and calculate the multiplicity list according to the transition probabilities.

Algorithm 9 Unbiased Partial Neighbor Search without Rejection-Free technique

select $\mathcal{N}_i$ for $i = 0, 1, \ldots, \mathcal{I} - 1$ where $\cup_{i=0}^{\mathcal{I}-1} \mathcal{N}_i(X) = \mathcal{N}(X)$
initialize $i = 0$ ▷ start with neighbor set $\mathcal{N}_0$
initialize $L = L_0$ ▷ start with L_0 remaining samples
initialize X_0 ▷ initial the starting state
for k in 1 to K **do**
  random $Y \in \mathcal{N}_i(J_{k-1})$ based on $\mathcal{Q}_i(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < \frac{\pi(Y)\mathcal{Q}_i(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}_i(X_{k-1}, Y)}$ **then**
    ▷ accept with probability $\min\left\{1, \frac{\pi(Y)\mathcal{Q}_i(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}_i(X_{k-1}, Y)}\right\}$
    $X_k = Y$ ▷ accept and move to state Y
  else
    $X_k = X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
  L = L - 1 ▷ a new sample from $\mathcal{N}_i$
  if $L = 0$ **then** ▷ if we don't have enough remaining samples
    $L = L_0$, and $i = i + 1 \mod \mathcal{I}$ ▷ switch to the next $\mathcal{N}_i$
  end if
end for

Proposition 4.5.1. *Suppose we have a state space $\mathcal{S}$, a reference measure μ, and a target density π, the proposal distribution $\mathcal{Q}$ and the corresponding neighbor set $\mathcal{N}$. In addition, suppose the Partial Neighbor Set $\{\mathcal{N}_i\}_{i=0}^{\mathcal{I}-1}$ satisfies all the conditions in Definition 4.5.1. Then $\pi(x)\mu(dx)$ is the stationary distribution for Algorithm 9 with the partial neighbor set $\mathcal{N}_i$.*

Proof. Let $\mathbf{P}_i(x, dy)$ be the transition probability for Partial Neighbor Set $\mathcal{N}_i$.

Then, $\forall y \in \mathcal{N}_i(x)$, we have

$$\begin{aligned}
\mathbf{P}_i(x, dy) &= \mathcal{Q}_i(x, y) \min\left(1, \frac{\pi(y)\mathcal{Q}_i(y, x)\mu(dy)}{\pi(x)\mathcal{Q}_i(x, y)\mu(dx)}\right) \\
&= \frac{q(x, y)}{\sum_{z \in \mathcal{N}_i(x)} q(x, z)} \min\left(1, \frac{\pi(y)\frac{q(y,x)}{\sum_{z \in \mathcal{N}_i(y)} q(y,z)}\mu(dy)}{\pi(x)\frac{q(x,y)}{\sum_{z \in \mathcal{N}_i(x)} q(x,z)}\mu(dx)}\right) \\
&= \min\left(\frac{q(x, y)}{\sum_{z \in \mathcal{N}_i(x)} q(x, z)}, \frac{\pi(y)\frac{q(y,x)}{\sum_{z \in \mathcal{N}_i(y)} q(y,z)}\mu(dy)}{\pi(x)\mu(dx)}\right)
\end{aligned} \tag{4.14}$$

Note that, $\forall y \in \mathcal{N}_i(x)$, we have $x \in \mathcal{N}_i(y)$. Thus, we have

$$
\begin{aligned}
&\pi(x)\mu(dx)\mathbf{P}_i(x,dy)\\
=&\pi(x)\mu(dx)\min\left(\frac{q(x,y)}{\sum_{z\in\mathcal{N}_i(x)}q(x,z)},\frac{\pi(y)\frac{q(y,x)}{\sum_{z\in\mathcal{N}_i(y)}q(y,z)}\mu(dy)}{\pi(x)\mu(dx)}\right)\\
=&\min\left(\pi(x)\frac{q(x,y)}{\sum_{z\in\mathcal{N}_i(x)}q(x,z)}\mu(dx),\pi(y)\frac{q(y,x)}{\sum_{z\in\mathcal{N}_i(y)}q(y,z)}\mu(dy)\right)\\
=&\pi(y)\mu(dy)\mathbf{P}_i(y,dx)
\end{aligned}
\tag{4.15}
$$

On the other hand, $\forall y\notin\mathcal{N}_i(x)$, $\pi(x)\mu(dx)\mathbf{P}_i(x,dy)=0=\pi(y)\mu(dy)\mathbf{P}_i(y,dx)$.
Thus, by reversibility, $\mathcal{N}_i$ is stationary with $\pi(x)\mu(dx)$. □

Proposition 4.5.2. *Suppose we have a state space $\mathcal{S}$, a reference measure μ, a target density π, and a Markov chain $\{X_0,X_1,X_2,\dots\}$ produced by algorithm 9. In addition, suppose $\pi(x)\mu(dx)$ is the stationary distribution for Algorithm 9 with all $\{\mathcal{N}_i\}_{i=0}^{\mathcal{I}-1}$, and $\bigcup_{i=0}^{\mathcal{I}-1}\mathcal{N}_i$ makes the Markov chain irreducible. Moreover, suppose there are rejections for the Markov chain, and thus the Markov chain is aperiodic. Then the Markov chain converges in total variation distance; i.e.:*

$$
\lim_{k\to\infty}\sup_{\mathcal{A}\subset\mathcal{S}}\left|\mathbf{P}(X_k\in\mathcal{A})-\int_{\mathcal{A}}\pi(y)\mu(dy)\right|=0 \tag{4.16}
$$

Proof. This follows immediately from Theorem 13.0.1 in [61]. □

We have proved the Markov chain Convergence Theorem (2.1.12) for discrete cases only in Section 2.1. This is the most fundamental theorem for the Markov chain, and thus we used a lot of space to review it. Here, we cited the Markov chain Convergence Theorem for general state space from [61]. To prove the theorem in general state space, it will need even much more steps than the discrete case, and thus we didn't review it in our thesis. Please see the proof of Theorem 13.0.1 in [61] for the details.

Theorem 4.5.3. *Suppose we have a state space $\mathcal{S}$, a reference measure μ, a target density π, a Markov chain $\{X_0,X_1,X_2,\dots\}$ produced by Algorithm 9, and a jump chain $\{(J_0,M_0),(J_1,M_1),(J_2,M_2),\dots\}$ produced by Algorithm 7. Meanwhile, suppose the proposal distribution $\mathcal{Q}$ and the corresponding neighbor set $\mathcal{N}$ ensure the Markov chain produced by the Metropolis-Hastings algorithm converges to the stationarity $\pi(x)\mu(dx)$. In addition, suppose $\pi(x)\mu(dx)$ is the stationary distribution for all $\{\mathcal{N}_i\}_{i=0}^{\mathcal{I}-1}$, and $\bigcup_{i=0}^{\mathcal{I}-1}\mathcal{N}_i$ makes both chains irreducible. Moreover, suppose both chains are aperiodic. Then the jump chain has the following properties:*

1. *The conditional distribution of M_k given J_k is equal to the distribution of $1+G$ where G is a geometric random variable with success probability $\alpha_i(J_k)$ where $\alpha_i(x)\coloneqq 1-P_i(x,\{x\})$, and P_i is the transition probability for partial neighbor set $\mathcal{N}_i$;*
2. *the transition probability $\hat{\mathbf{P}}_i$ from the jump chain satisfy $\hat{\mathbf{P}}_i(x,dy)=\frac{1}{\alpha_i(x)}\mathbf{P}_i(x,dy)\mathbb{1}(x\neq y)$, and $\hat{\mathbf{P}}_i(x,\{x\})=0$;*
3. *If the original chain is ϕ-irreducible (see, e.g., [61]) for some positive σ-finite measure ϕ on $\mathcal{S}$, then the jump chain is also ϕ-irreducible for the same ϕ.*

4. *If the Markov chain has stationary distribution $\pi(x)\mu(dx)$ given $\mathcal{N}_i$, then the jump chain has stationary distribution $\hat{\pi}(x) = c_i\alpha_i(x)\pi(x)\mu(dx)$ where $c_i^{-1} = \int \alpha_i(y)\pi(y)\mu(dy)$*

5. *If $h : \mathcal{S} \to \mathbb{R}$ has finite expectation, then with probability 1,*

$$\lim_{K\to\infty} \frac{\sum_{k=1}^{K} M_k h(J_k)}{\sum_{k=1}^{K} M_k} = \pi(h) := \int h(x)\pi(x)\mu(dx).$$

Proof. The proof is trivial given the Proposition 4.5.2 and Theorem 3.3.6. □

4.6 QUBO Revisited: Choices for the Partial Neighbors

4.6.1 Systematic Or Random

In the section 4.3, we used two systematically pre-selected neighbor sets $\mathcal{N}_0$, $\mathcal{N}_1$. However, there can be various ways to select the Partial Neighbor Sets, all of which, as long as the Partial Neighbor selected satisfies the PNS Convergence Theorem 4.5.3, will converge to the target density π. To start with, we compare two ways of choosing partial neighbor sets here: systematic and random. For simplicity, assume that we have N neighbors for all states, and we use partial neighbor sets of size n. Therefore, we have $\binom{N}{n}$ different partial neighbor sets. For systematic method, we choose $\mathcal{I}$ partial neighbor sets $\{\mathcal{N}_i\}_{i=1}^{\mathcal{I}}$, where $\cup_{i=1}^{\mathcal{I}}\mathcal{N}_i(x) = \mathcal{N}(x)$. We proceed with each Partial Neighbor Set for L_0 original samples and then move on to the next until we reach the $\mathcal{I}$-th one and then go back to the first one. We use the notation $\mathcal{N}_i(x)$ for systematic Partial Neighbor Sets because $\mathcal{N}_i(x)$ is pre-determined for $i = 1, 2, \ldots, \mathcal{I}$. On the other hand, for random Partial Neighbor Sets, we choose a new set $\mathcal{N}_k$ from all $\binom{N}{n}$ potential Partial Neighbor Sets after each L_0 original samples. We use the notation $\mathcal{N}_k(x)$ for random partial neighbor sets because $\mathcal{N}_k(x)$ can be different for every PNS step, and the subscript k represents the special partial neighbor set for step k. For both methods, $\mathcal{Q}_i(x, y), \mathcal{Q}_k(x, y) \propto \mathcal{Q}(x, y)$ for $y \in \mathcal{N}_i(x)$, and $\mathcal{Q}_i(x, y) = \mathcal{Q}_k(x, y) = 0$ otherwise.

To compare the above two methods of selecting Partial Neighbor Sets, we apply them to the previous 16×16 QUBO question, and we test the following four scenarios:

1. two systematic partial neighbor sets where the first set considers flipping the first half of the bits, and the second set considers flipping the second half of the bits;

2. four systematic partial neighbor sets where each set considers flipping a quarter of the bits;

3. random partial neighbor sets with $\frac{N}{2}$ partial neighbors; that is, each set considers flipping a random set of bits with size $\frac{N}{2}$;

4. random partial neighbor sets with $\frac{N}{4}$ partial neighbors; that is, each set considers flipping a random set of bits with size $\frac{N}{4}$;

The result is shown in Figure 4.8; for this case, systematic Partial Neighbor Sets are better than random Partial Neighbor Sets. However, random Partial Neighbor Sets can be better when we run the same code with a different random seed. After running this simulation for 100 different random seeds, the systematic neighbor sets are better 56 times. Thus, we conclude that the performance of these two Partial Neighbor Sets is close to each other. We will continue using the systematic Partial Neighbor Sets in our later simulation.

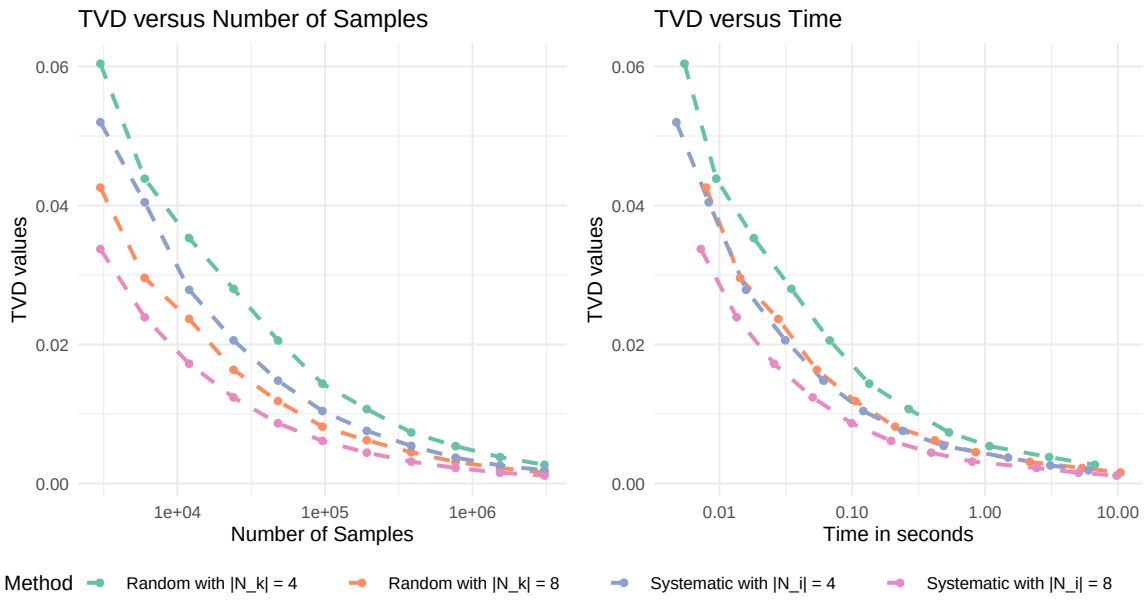

Figure 4.8: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for four scenarios: Systematic PNS and Random PNS, each with Partial Neighbor Set sizes of 4 and 8. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. For all PNS, we used $L_0 = 100$.

4.6.2 The choice of the Partial Neighbor Sets sizes

In previous simulations, we naively use we used $|\mathcal{N}_i(x)| = 4$ or 8 and $L_0 = 100$ in previous examples. What is the optimal choice for $|\mathcal{N}_i(x)|$? We want to compare $|\mathcal{N}_i(x)| = 2, 4, 6, 8, \ldots, 14$ by the QUBO question. In previous cases, we only used the systematic Partial Neighbor Set size n that can be divided evenly by N. For other n's that cannot be divided evenly such as 14, we create the Partial Neighbor Sets in loops to make the sets have the same size and include all entries for the same amount of time. For example, we create the following 8 Partial Neighbor Sets for $|\mathcal{N}_i(x)| = 14$:

- $\mathcal{N}_1(x) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$; that is, we consider flipping entries 1 to 14;
- $\mathcal{N}_2(x) = \{15, 16, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$;
- $\mathcal{N}_3(x) = \{13, 14, 15, 16, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$;
- $\mathcal{N}_4(x) = \{11, 12, 13, 14, 15, 16, 1, 2, 3, 4, 5, 6, 7, 8\}$;
- $\mathcal{N}_5(x) = \{9, 10, 11, 12, 13, 14, 15, 16, 1, 2, 3, 4, 5, 6\}$;
- $\mathcal{N}_6(x) = \{7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1, 2, 3, 4\}$;
- $\mathcal{N}_7(x) = \{5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1, 2\}$;
- $\mathcal{N}_8(x) = \{3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$;

Figure 4.9 shows the results for comparing the Unbiased PNS with $|\mathcal{N}_i| \equiv 2$, 4, 6, 8, 10, 12, and 14 for $\forall X \in \{0, 1\}^{16}$. Every other simulation setting is the same as the previous simulations for the QUBO question. The choice of L_0 is still 100. According to the left plot, we can say that given the same amount of original samples, the Markov chain from $|\mathcal{N}_i| = 14$ is the least biased. On the other hand, from the right plot, we can conclude that, given the same amount of CPU time, the sample quality from $|\mathcal{N}_i| = 14$ is the best. In addition, the performances are close to each other for all cases where $|\mathcal{N}_i| \geq 8$. Note that a single-core implementation makes all these comparisons by the CPU time, and parallel hardware can provide speedups. Intuitively, the more tasks that can be calculated simultaneously, the greater the speedup. Thus, if we apply our Unbiased PNS on parallel hardware with a limited number of parallel tasks that can be computed simultaneously, we should choose the largest possible partial neighbor set size $|\mathcal{N}_i|$.

4.6.3 The choice of L_0

Furthermore, Figure 4.10 shows the results for comparing the Unbiased PNS with $L_0 = 10$, 50, 100, 500, and 1000. Again, every other setting of the simulation is the same, and $|\mathcal{N}_i|$ is still 8, $\forall x \in \{0, 1\}^{16}$. The left plot shows that given the same samples, the Markov chain from $L_0 = 10$ is the least biased. However, the right plot shows that, given the same amount of CPU time, the TVD values are about the same except $L_0 = 10$. The case with $L_0 = 100$ is slightly better than the other cases, but the difference is not too large. $L_0 = 10$ becomes the worst since such L_0 has too many rejections (about one rejection for every ten samples). Thus, for a single-core implementation, the choice of L_0 is not that important as long as it is not extreme. In addition, for parallel hardware, when we change the partial neighbors being chosen, we have to bring the new neighbors to the memory. This can be a waste of time if we are doing this very frequently. Thus, L_0 should not be too small for parallel computing.

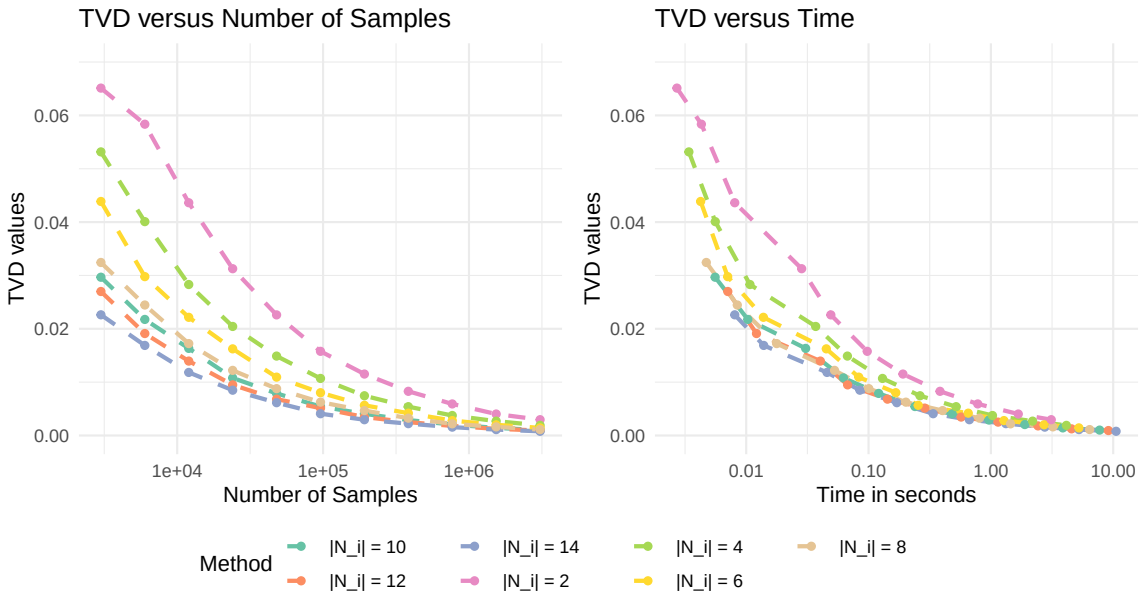

Figure 4.9: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for four scenarios: Unbiased PNS with different partial neighbor set sizes $\{2, 4, 6, \ldots, 14\}$. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. For all PNS, we used $L_0 = 100$.

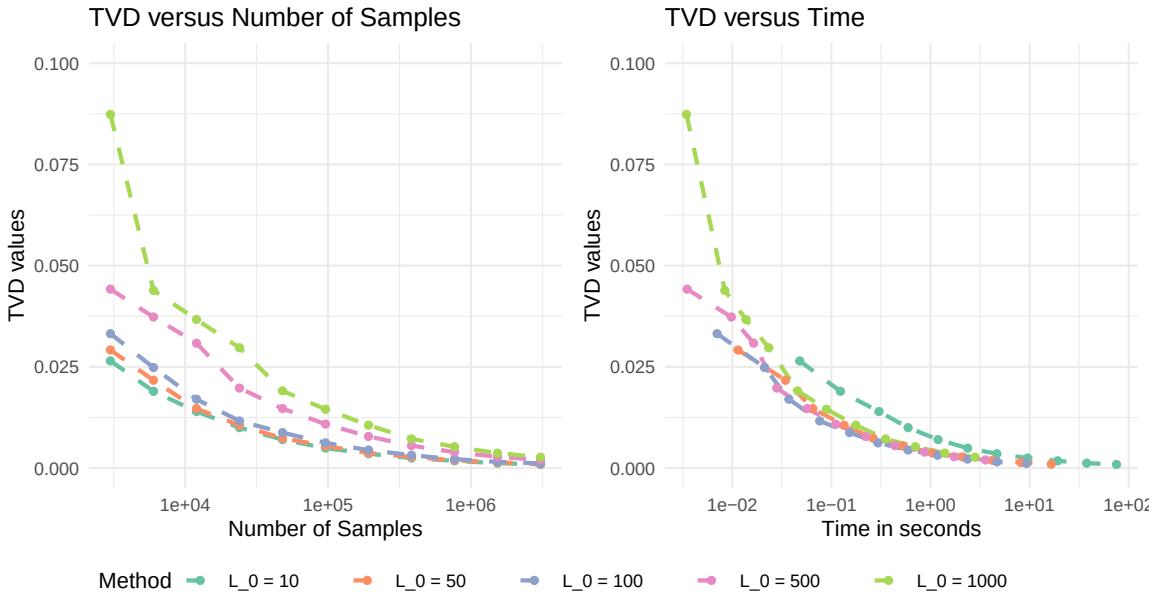

Figure 4.10: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased Partial Neighbor Search with different sizes of L_0. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{300, 600, 1200, 2400, \ldots, 3072000\}$. For all PNS, we used $|\mathcal{N}_i| = 8$.

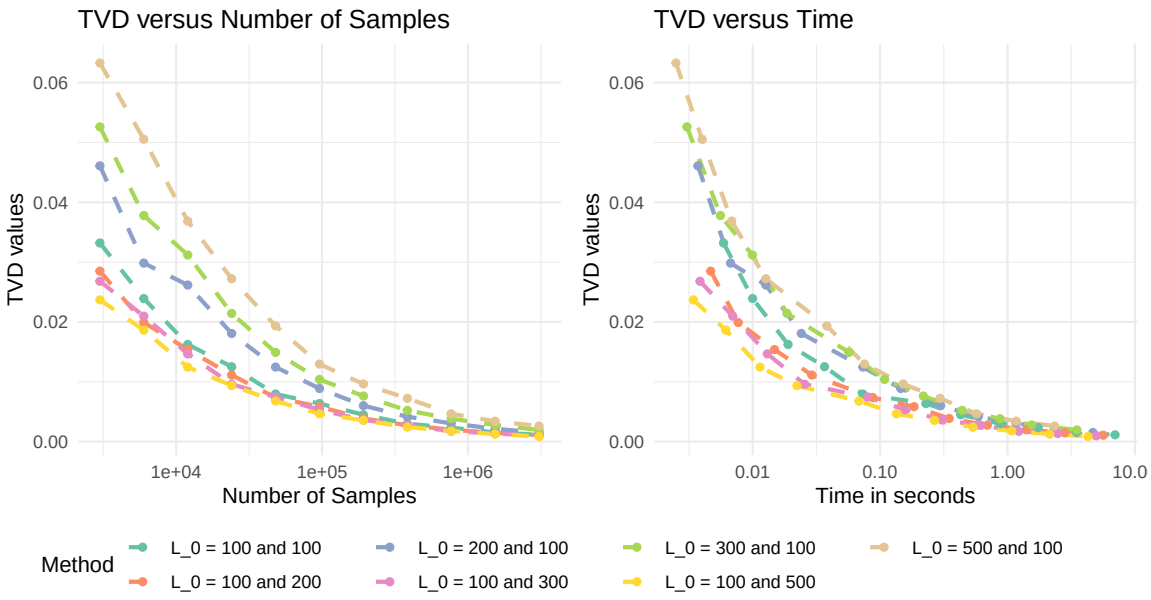

Figure 4.11: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with different L_0 values chosen for two partial neighbor sets. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 300 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. For all PNS, we used two partial neighbor sets, where each of them considers flipping 8 entries.

4.6.4 The choice of $\mathbf{L_0}$ when the Partial Neighbor Sets have different sizes

In the previous section, we found the best choices for L_0 and Partial Neighbor Sets sizes by controlling the other parameter, and we also used Partial Neighbor Sets of the same size. Here, we try to explore the sampling efficiency with Partial Neighbor Sets of different sizes, and the best choice of the corresponding $\mathbf{L_0}$. Since Partial Neighbor Sets have different sizes, $\mathbf{L_0}$ is a vector with a size equal to the total number of all Partial Neighbor Sets. We only use two Partial Neighbor Sets $\mathcal{N}_1$ and $\mathcal{N}_2$ here. where $|\mathcal{N}_1| = n$, and $|\mathcal{N}_2| = N - n$. We can choose different $\mathbf{L_0}$ for different Partial Neighbor Sets. For example, we first let $n = 8$, then the $|\mathcal{N}_1| = |\mathcal{N}_2| = 8$, and we check the sampling efficiency when $\mathbf{L_0}(\mathcal{N}_1) = 100$ and $\mathbf{L_0}(\mathcal{N}_2) = 100$. We also check the combinations for $\mathbf{L_0}$ like $\{100, 10\}$, $\{100, 50\}$, $\{200, 100\}$, etc. The result is shown in Figure 4.11. Similarly, we use $n = 4$, so $|\mathcal{N}_1| = 4$ and $|\mathcal{N}_2| = 12$, and do the same simulation again. The results are shown in Figure 4.12.

First of all, Figure 4.12 confirms that the Markov chain will converge to the correct stationarity given that the sizes of the Partial Neighbor Sets are different. In addition, by Figure 4.11 and Figure 4.12, the best strategies are both using $L_0 = 100$ and 500 accordingly for $\mathcal{N}_1$ and $\mathcal{N}_2$. However, logically, this is not true for all QUBO questions, and this result only occurs because of the randomness of the QUBO matrix Q.

To eliminate the influence from the randomness of the QUBO matrix Q, we create one thousand QUBO matrices Q. For each of these QUBO matrices Q, we compare the methods of choosing L_0 by 100 runs, and we average the results from these 1000×100 simulations to get the average TVD

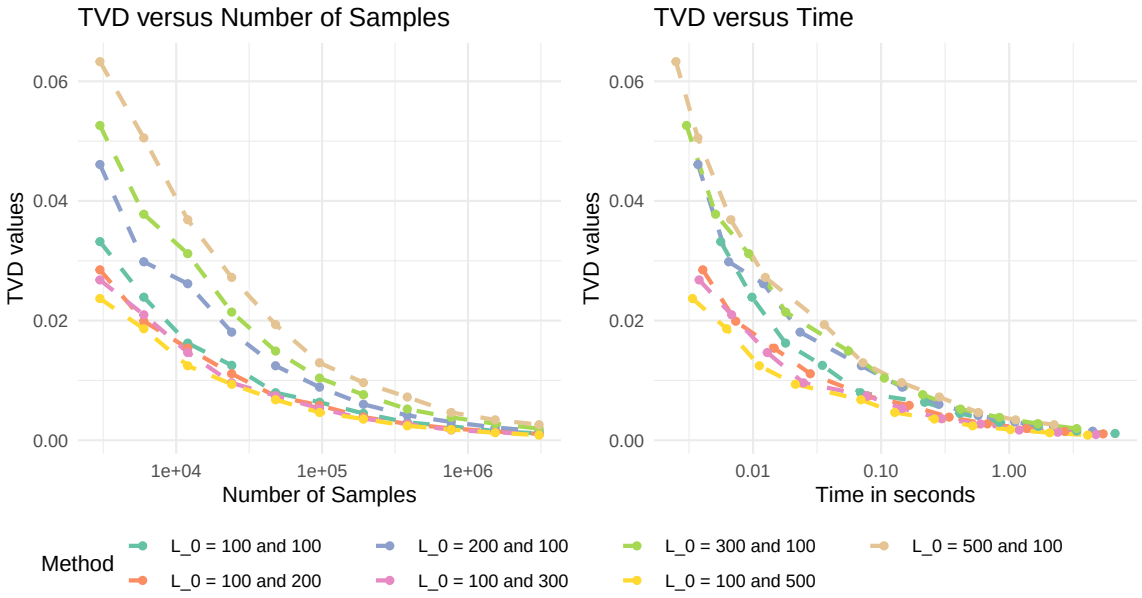

Figure 4.12: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with the sizes for Partial Neighbor Sets being 4 and 12. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 300 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$.

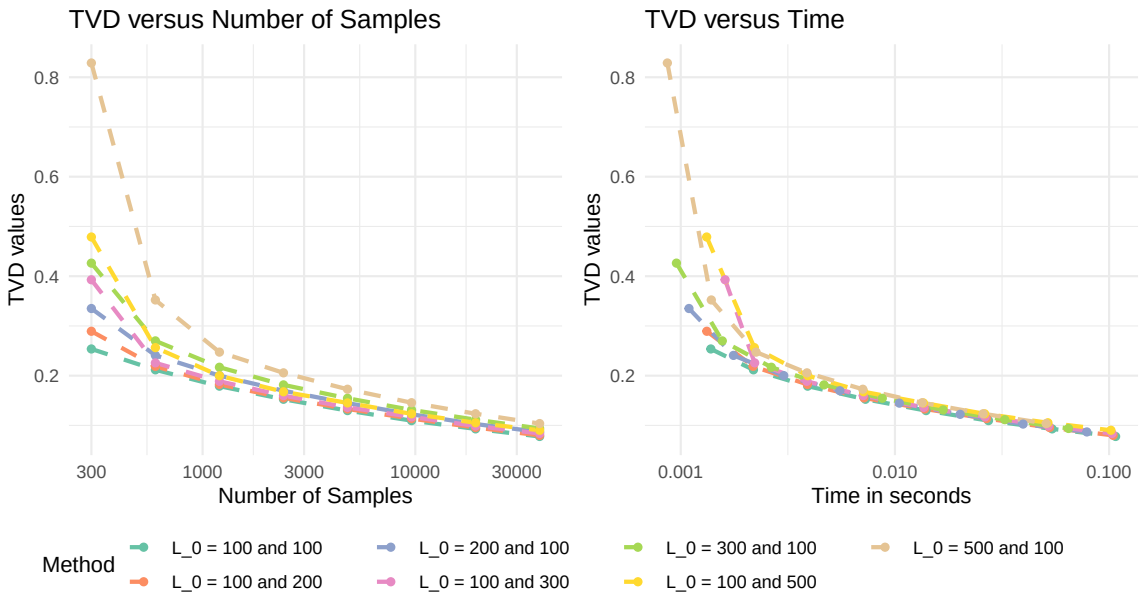

Figure 4.13: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with the sizes for Partial Neighbor Sets being 2 and 6. Random upper triangular 8×8 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 different QUBO matrix Q, and for each QUBO matrix Q, we did 100 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, 48000, 96000\}$.

values as well as the average time in seconds used by the program. Since this simulation has much more simulation runs than before, we use an 8×8 QUBO matrix here, and the sizes for Partial Neighbor Sets are 2 and 6. The result is shown in Figure 4.13, and we can conclude that when comparing the TVD values by time in seconds, the choice of L_0 does not matter too much even when the sizes for the Partial Neighbor Sets are different.

4.6.5 The choice of Partial Neighbor Sets sizes given $\mathbf{L_0}$

We compared the sampling efficiency with different combinations for $\mathbf{L_0}$ when the Partial Neighbor Sets have different sizes in Section 4.6.4. On the other hand, if we have a vector $\mathbf{L_0}$ for different numbers of samples with respect to the original chain, what will be the best choice for the Partial Neighbor Set sizes? For both $\mathbf{L_0} = \{100, 100\}$ and $\mathbf{L_0} = \{100, 300\}$, we check the sampling efficiency for different combinations of the sizes for two Partial Neighbor Sets $\mathcal{N}_1$ and $\mathcal{N}_2$. The result is shown in Figure 4.14. In addition, the results for $L_0 = \{100, 300\}$ is shown in Figure 4.15.

From Figure 4.14 and Figure 4.15, the results are similar to what we had in Section 4.6.4. We conclude that, for given sizes L_0, the choice of PNS sets does not make a big difference. Similar to Section 4.6.4, we can also try to eliminate the influence of the QUBO matrix Q. Again, we create one thousand QUBO matrices Q, and for each of these QUBO matrices Q, we compare the methods of choosing Partial Neighbor Sets by 100 runs. We average the results from these 1000×100 simulations to get the average TVD values as well as the average time in seconds used by the program. Again,

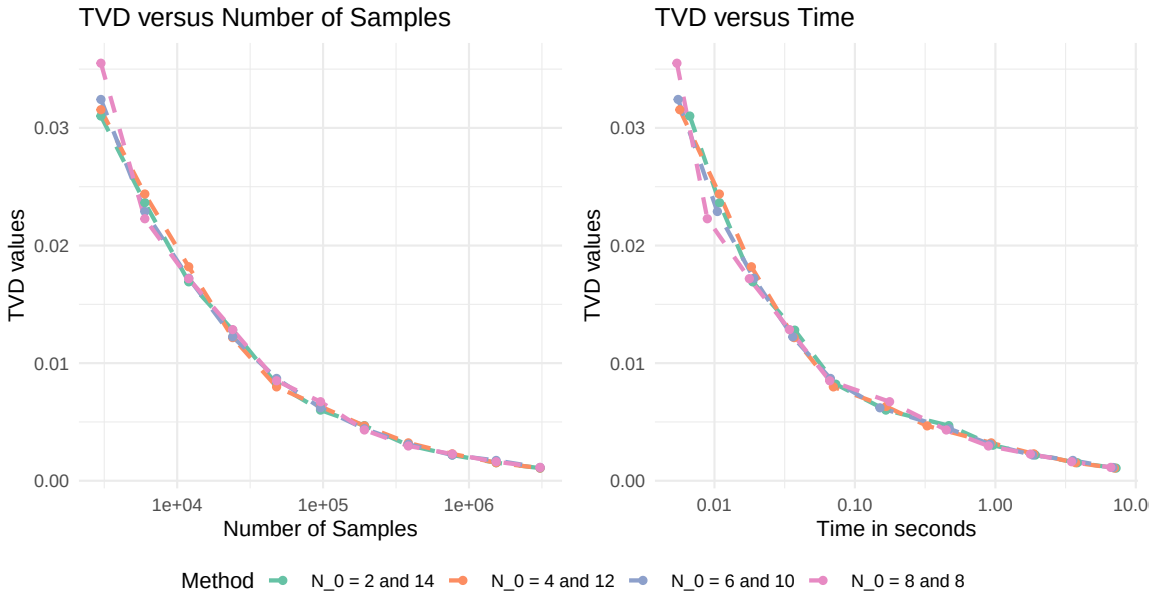

Figure 4.14: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with $\mathbf{L_0} = \{100, 100\}$. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$.

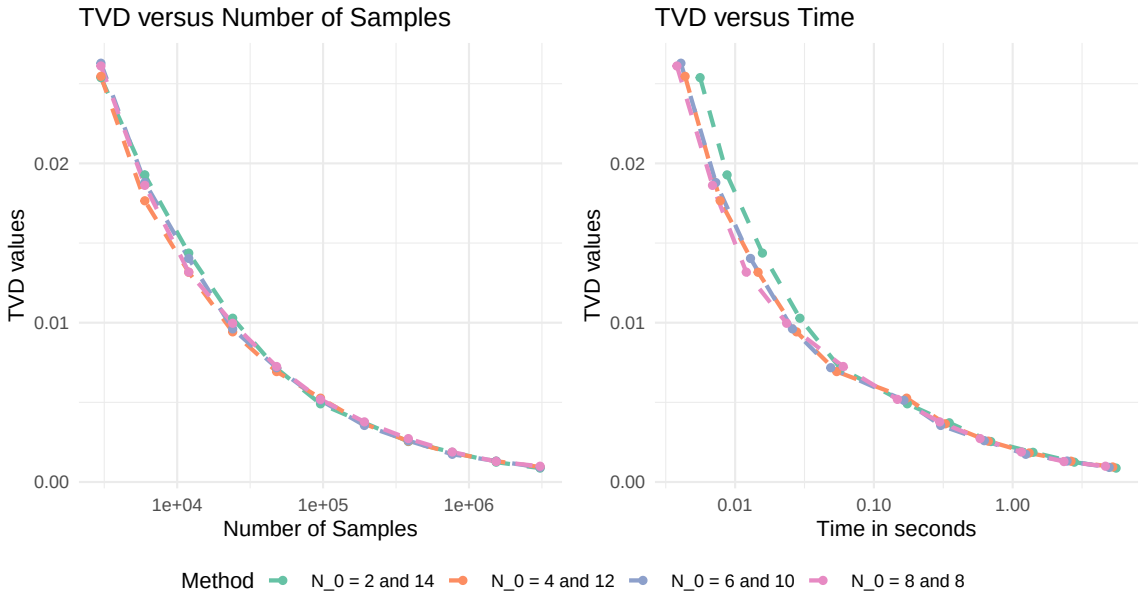

Figure 4.15: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with $\mathbf{L_0} = \{100, 300\}$. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$.

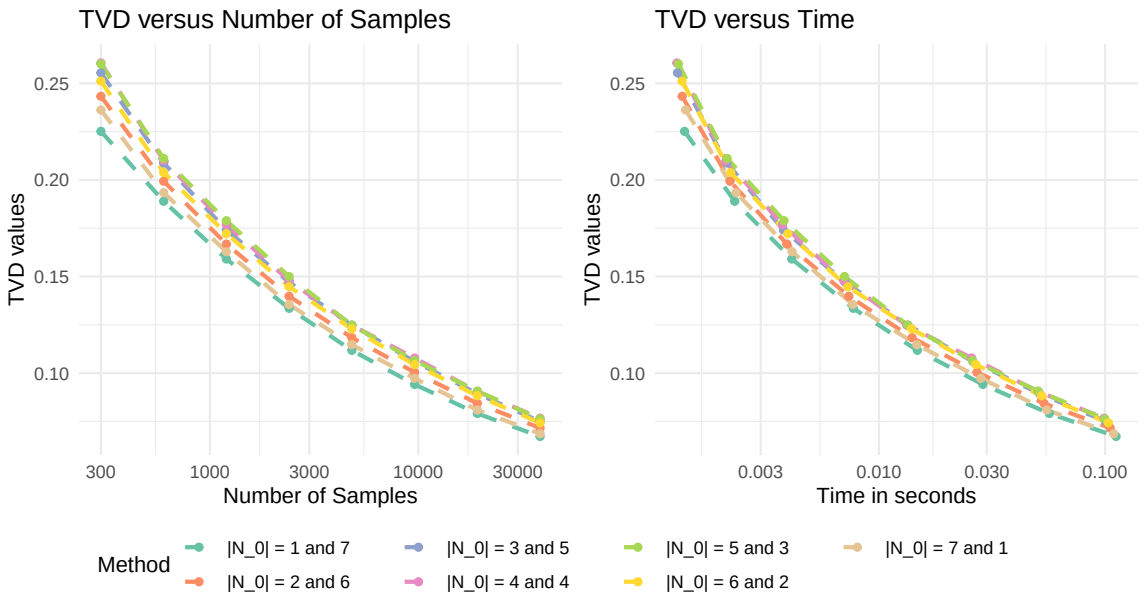

Figure 4.16: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with $L_0 = 100$ for both Partial Neighbor Sets. Random upper triangular 8×8 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 different QUBO matrix Q, and for each QUBO matrix Q, we did 100 simulation runs given a certain original sample size, where the sizes are $\{300, 600, 1200, 2400, 4800, 9600, 19200, 38400\}$.

we use an 8×8 QUBO matrix here, and the size for L_0 is 100 for both Partial Neighbor Sets. The result is shown in Figure 4.16, and we can conclude that when comparing the TVD values by time in seconds, the choice of Partial Neighbor Sets does not matter too as well.

4.6.6 Two Flips

In previous simulations, we used uniform proposal distributions among all neighbors where the neighbors are defined as binary vectors with Hamming distance 1. That is, $\mathcal{Q}(x, y) = \frac{1}{N}$ for $\forall y$ such that $|x-y| = \sum_{i=1}^{N} |x_i - y_i| = 1$, $\forall x, y \in \{0, 1\}^N$. Thus, the neighbors are all binary vectors different by one flip. We usually did not include two flips because of two reasons. First of all, the acceptance rate by the Metropolis-Hastings algorithm on two flips is low, since two flips usually have larger energy differences than one flip. In addition, there are too many of them. For example, for a 16×16 QUBO question, if we only consider one flip, then for each state, there are 16 neighbors. However, considering two flips, there will be 256 neighbors. Then for Rejection-Free, the number of neighbors is too many to be applied efficiently.

However, with the help of PNS, we can now try to include two flips into the Partial Neighbor Sets. For the first simulation, we use the same settings as before, except we choose the following proposal distribution and neighbors:

1. One Flip with four neighbors chosen randomly. That is, for each step, we randomly choose four entries from 1 to 16, and apply Partial Neighbor Search on these four neighbors.

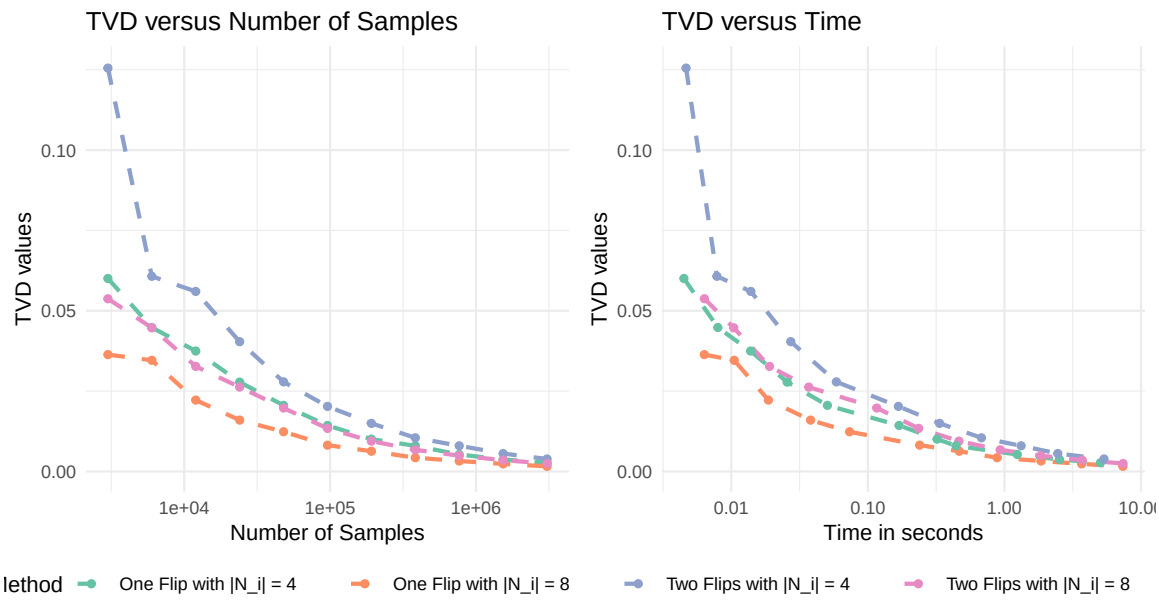

Figure 4.17: Two-flip: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with different selections of neighbor sets. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. $L_0 = 100$.

2. One Flip with eight neighbors chosen randomly.

3. Two Flip with four neighbors chosen randomly. We first randomly choose eight entries from 1 to 16, and apply Partial Neighbor Search on these four neighbors, to make one flip with four neighbors first for L_0 original samples. Then we randomly choose four groups of two different entries from 1 to 16 and apply Partial Neighbor Search on these four groups of two flips.

4. Two Flip with eight neighbors chosen randomly.

Note that, we can not only apply the pure two flips since the Markov chain will become reducible then, so we mix the one flip and two flips to make a hybrid chain. The results for the simulations are shown in Figure 4.17. Note that, the mean for the multiplicity list M_k (before considering L_0) is 7×10^{13}, which is super large. This is understandable since the energy difference for two flips is large. As a result, the two flip algorithms sample less efficiently than one flip. In addition, we did another simulation by generating $Q_{i,j} \sim N(0, 0.4^2)$ instead of using the standard deviation of 10. The simulation result is shown in Figure 4.18. It seems that even if the energy difference for two flips is not that large, two flips are still a little bit worse than one flip. Technically, we think two flips can be very helpful when escaping from the local mode, but in our simulation, we didn't find it to be more efficient.

In addition to the previous comparison, we have more ways to apply two flips. We can define the neighbors to state with Hamming distance less or equal to 2. That is, $\mathcal{Q}(x, y) = \frac{1}{N^2}$ for $\forall y$ such that $|x - y| = \sum_{i=1}^{N} |x_i - y_i| \leq 2$, $\forall x, y \in \{0, 1\}^N$. Then we choose $|\mathcal{N}_i|$ neighbors as Partial Neighbor

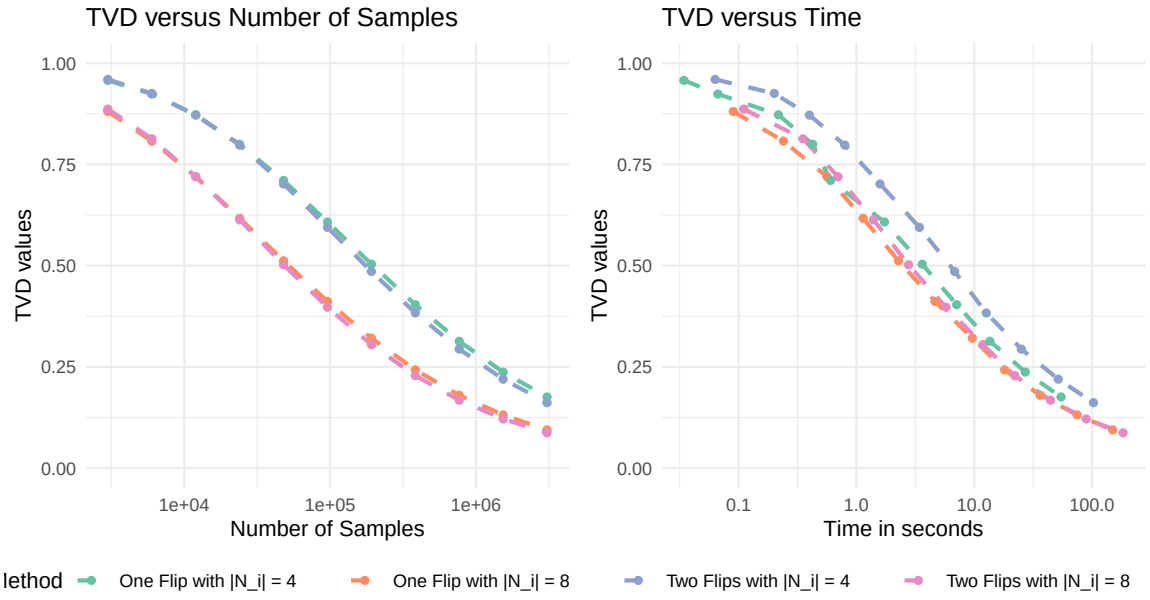

Figure 4.18: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with different selections of neighbor sets: one flip and two flips. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 0.4^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. $L_0 = 100$.

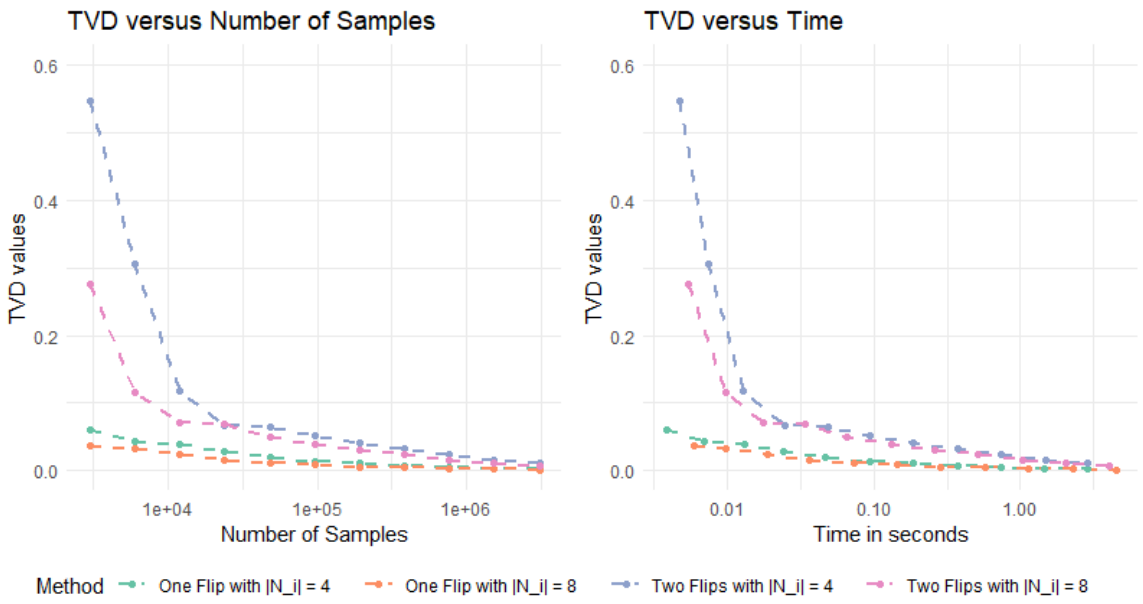

Figure 4.19: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with different selections of neighbor sets: one flip and two flips. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 0.4^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. $L_0 = 100$.

Set randomly from all N^2 neighbors. The result of the comparison is shown in Figure 4.19. Again, the two flips are worse than one flip. This is easy to understand as well since most of the two flips have higher energy differences, and thus the two flips are not making as much contribution as one flip when being included in the Partial Neighbor Sets. Moreover, we can try to adjust the proposal distribution $\mathcal{Q}$ to make our Partial Neighbor Sets consider one flip more. We choose $\mathcal{Q}(x, y) = \frac{1}{2N}$ for $|x-y| = \sum_{i=1}^{N} |x_i - y_i| = 1$ and $\mathcal{Q}(x, y) = \frac{1}{2(N^2-N)}$ for $|x-y| = \sum_{i=1}^{N} |x_i - y_i| = 2$, and the result is shown in Figure 4.20. The difference between one flip and two flips is not as much as before, but two flips are still worse than one flip. Thus, here we conclude that for the situation we have here, the two flips may not be that useful.

4.7 Use the information of QUBO matrix

We already talked about using different proposal distributions other than a uniform proposal distribution on all neighbors in Section 4.6.6. We believe that uniform proposal distribution can be a good choice generally, but when we are given some extra information about the sampling question, there must exist some other efficient ways to create better proposal distribution. For example, for the QUBO question, maybe we can make use of the information of the QUBO matrix Q to get a

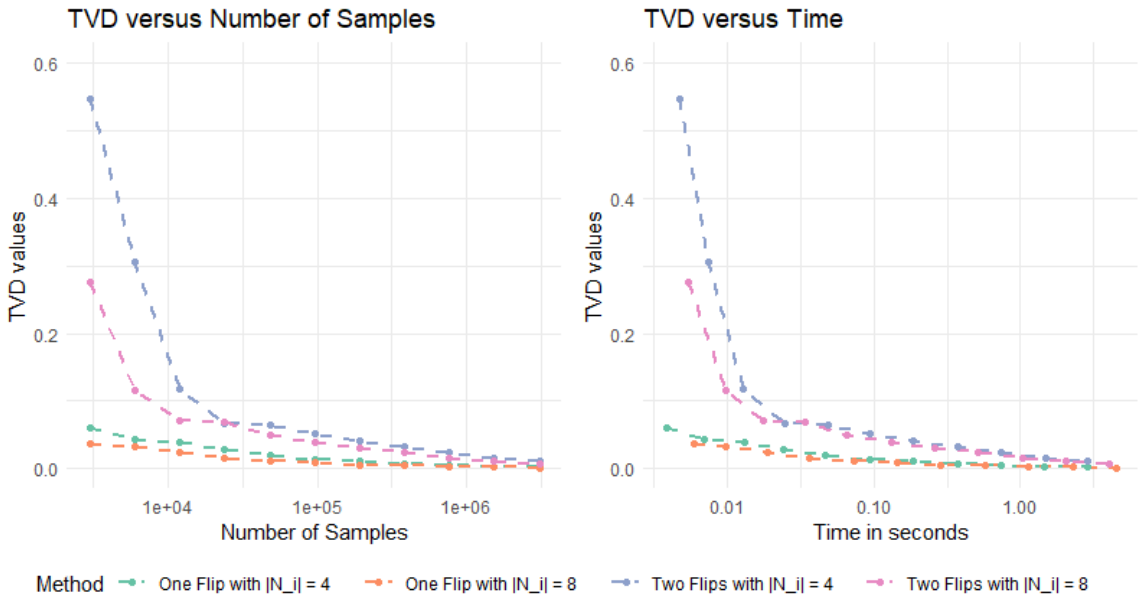

Figure 4.20: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with different selections of neighbor sets: one flip and two flips. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 0.4^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 1000 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$. $L_0 = 100$.

better proposal distribution. First, we define a parameter weight w_i for each entry of x_i to be:

$$w_i \propto \sum_{j=1}^{N} [|Q_{i,j}| + |Q_{j,i}|] - |Q_{i,i}| \tag{4.17}$$

Then, based on the vector $\mathbf{w}$, we consider four types of Modified PNS

1. $\mathcal{Q}(X,Y) \propto \frac{1}{w_i}$ if $|X_i - Y_i| = 1$
2. $\mathcal{Q}(X,Y) \propto \frac{1}{w_i^2}$ if $|X_i - Y_i| = 1$
3. $\mathcal{Q}(X,Y) \propto w_i$ if $|X_i - Y_i| = 1$
4. $\mathcal{Q}(X,Y) \propto w_i^2$ if $|X_i - Y_i| = 1$

The result of the simulation is shown in Figure 4.21. From the top two figures, we can see that Modified 3 and Modified 4 are less efficient. To see the difference between regular PNS, Modified 1, and Modified 2, we make two more plots for them on a smaller scale as the bottom two figures. From these two figures, we can see that, for this circumstance, the Modified PNS with proposal distribution being propositional to the inverse of the weights is better than the regular PNS. This is understandable, since usually for sampling questions, we care more about the states around the highest density values. For example, if we are sampling from space with only four density values $\{100, 99, 0.1, 0.09\}$, then the two small states do not matter too much. To get better samples, we probably want our Markov chain to be switching between state 100 and 99. A similar thing can happen here, for this QUBO question, we want our Markov chain to switch among states with large density values, and those large density values states are different by the flips with less weight. Therefore, the Modified versions of PNS can help.

Although the Modified PNS is better when we used a randomly generated QUBO matrix Q, there can be some other problems for some special Q. For example, we define the following 4 by 4 QUBO matrix Q:

$$\begin{pmatrix} 2.5 & 3 & -5.0 & -6.0 \\ 0.0 & 4 & -6.0 & -7.0 \\ 0.0 & 0 & 5.5 & 6.5 \\ 0.0 & 0 & 0.0 & 7.0 \end{pmatrix} \tag{4.18}$$

For target density from this QUBO matrix, there are two separate local maximum $(1,1,0,0)^T$ and $(0,0,1,1)^T$. Then, in this case, we probably don't want to be stuck at one local maximum and thus Modified 1 and 2 may not be that good. Note that, the weights are $\{0.4232, 0.461, 0.077, 0.038\}$ accordingly.

The result of the simulation is shown in Figure 4.22. From the figures, Modified 1 and 2 are a little bit worse than the regular PNS. This shows that the Modified version of the PNS is not always better than the regular PNS. However, we still believe that the Modified PNS can be useful in most unimodal cases.

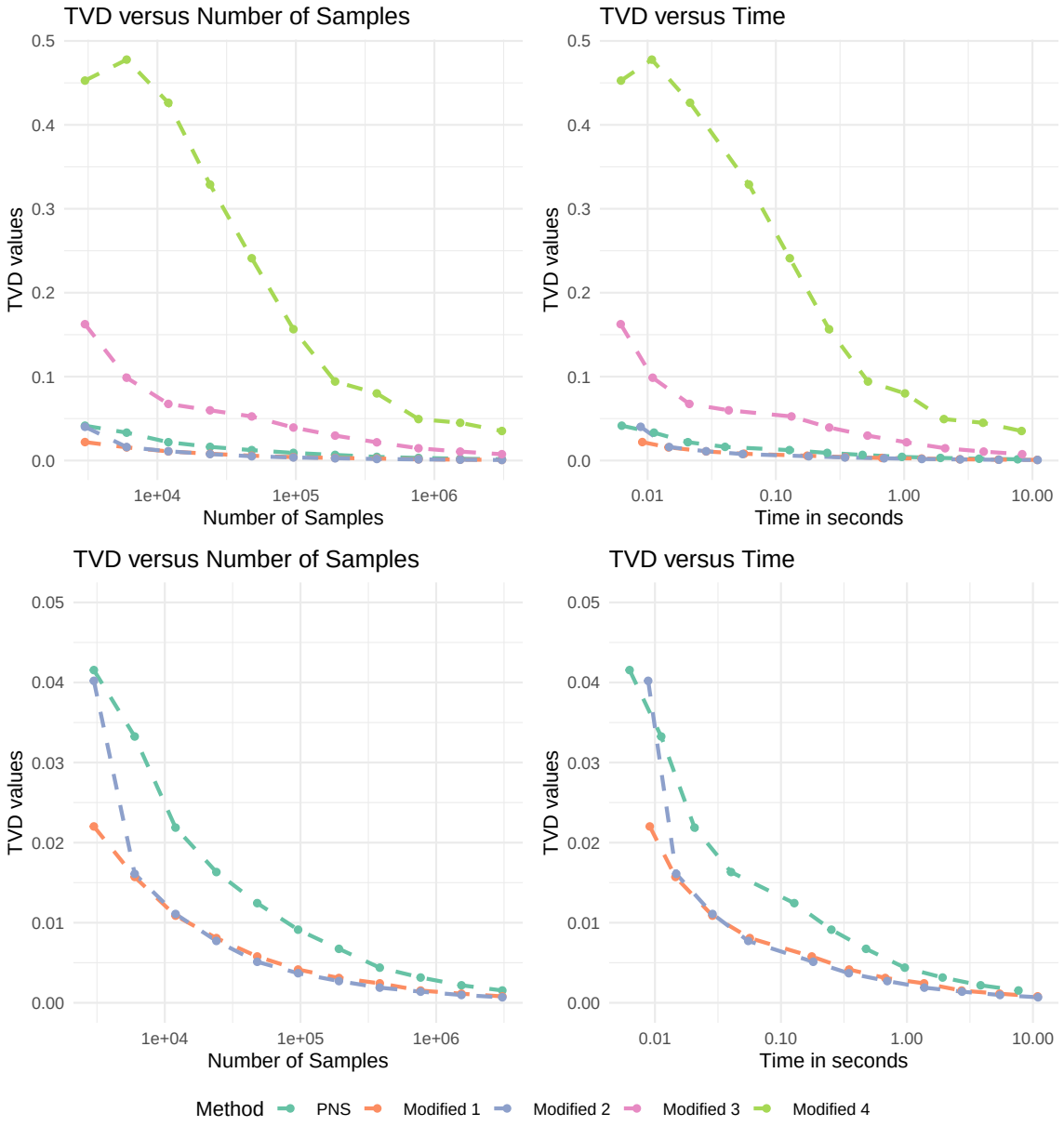

Figure 4.21: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with and without using the information of matrix Q. PNS Modified means proposal distribution based on the QUBO matrix Q as $w_i \propto \sum_{j=1}^{N}[|Q_{i,j}|+|Q_{j,i}|]-|Q_{i,i}|$. The upper two plots are the same as the bottom two plots except for the scales. Modified 1: $\mathcal{Q}(X,Y) \propto \frac{1}{w_i}$ if $|X_i - Y_i| = 1$; Modified 2: $\mathcal{Q}(X,Y) \propto \frac{1}{w_i^2}$ if $|X_i - Y_i| = 1$; Modified 3 : $\mathcal{Q}(X,Y) \propto w_i$ if $|X_i - Y_i| = 1$; Modified 4: $\mathcal{Q}(X,Y) \propto w_i^2$ if $|X_i - Y_i| = 1$. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 10^2)$ for upper triangular elements. Each dot within the plot represents the average TVD value and time used for 100 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$.

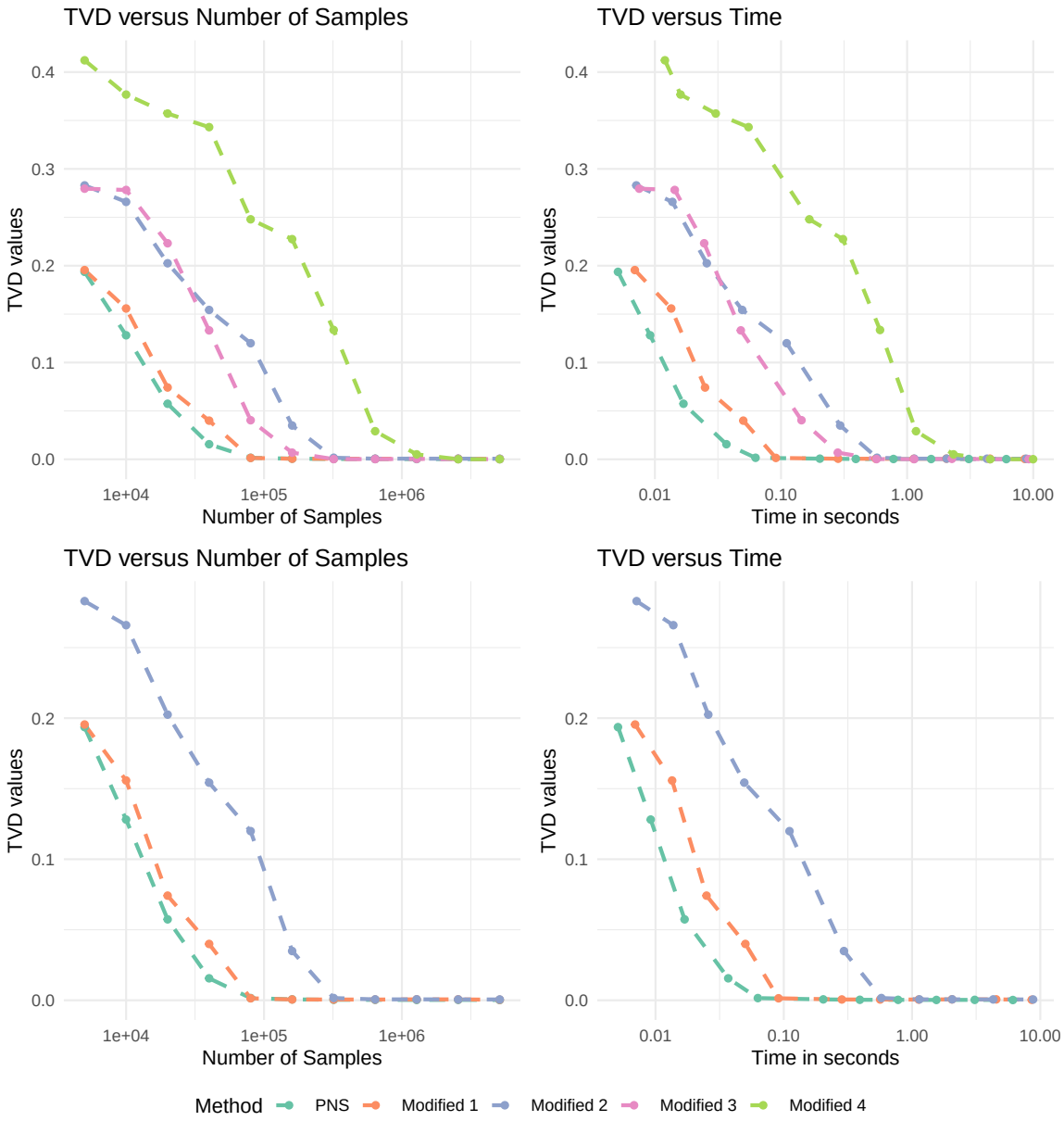

Figure 4.22: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for Unbiased PNS with and without using the information of matrix Q. PNS Modified means proposal distribution based on the QUBO matrix Q as $w_i \propto \sum_{j=1}^{N}[|Q_{i,j}|+|Q_{j,i}|]-|Q_{i,i}|$. The upper two plots are the same as the bottom two plots except for the scales. Modified 1: $\mathcal{Q}(X,Y) \propto \frac{1}{w_i}$ if $|X_i - Y_i| = 1$; Modified 2: $\mathcal{Q}(X,Y) \propto \frac{1}{w_i^2}$ if $|X_i - Y_i| = 1$; Modified 3 : $\mathcal{Q}(X,Y) \propto w_i$ if $|X_i - Y_i| = 1$; Modified 4: $\mathcal{Q}(X,Y) \propto w_i^2$ if $|X_i - Y_i| = 1$. Random upper triangular 4×4 given QUBO matrix. Each dot within the plot represents the average TVD value and time used for 100 simulation runs given a certain original sample size, where the sizes are $\{3000, 6000, 12000, 24000, \ldots, 3072000\}$.

4.8 Irreversible Assumptions for Partial Neighbor Sets

In section 4.4, we compared PNS with the Metropolis-Hastings algorithm by a two-dimensional donut example. In that example, we used a reversible proposal. That is, for Partial Neighbor Set $\mathcal{N}_i$, $y \in \mathcal{N}_i(x) \iff x \in \mathcal{N}_i(y)$, $\forall x, y \in \mathcal{S}$. Thus, we chose the Partial neighbor Set $\mathcal{N}_i(x)$ as follows:

1. generate $\delta_1, \delta_2 \sim \text{Normal}(0, 1)$;
2. for state $x = (x_1, x_2)$, put $y = (x_1 + \delta_1, x_2 + \delta_2)$ into the Partial Neighbor Set $\mathcal{N}_i(x)$;
3. to ensure the reversibility, also put $y' = (x_1 - \delta_1, x_2 - \delta_2)$ into the Partial Neighbor Set $\mathcal{N}_i(x)$;
4. repeat the above steps 25 times to generate a total of 50 neighbors for the Partial Neighbor Set $\mathcal{N}_i(x)$.

In Section 4.5, we proved that this reversible PNS chain will converge to the target density correctly. However, is reversibility necessary for the Partial Neighbor Set?

We use a four-dimension of donuts example to show the results from a PNS chain with irreversible Partial Neighbor Sets. Suppose we have four independent random variables μ, θ_1, θ_2, θ_3 where

$$\mu \sim \text{Normal}^+(\mu_0, \sigma^2),\ \theta_1, \theta_2, \theta_3 \sim \text{Uniform}[0, \pi). \tag{4.19}$$

Here, Normal^+ means the Truncated Normal distribution without the negative tail, and π in the Uniform distribution means the circular constant instead of the target density. Then we define two random variables X_1 and X_2 to be

$$\begin{aligned} X_1 &= \sqrt{\mu}\sin\theta_1 \sin\theta_2,\ X_2 = \sqrt{\mu}\sin\theta_1\cos\theta_2, \\ X_3 &= \sqrt{\mu}\cos\theta_1 \sin\theta_3,\ X_4 = \sqrt{\mu}\cos\theta_1\cos\theta_3. \end{aligned} \tag{4.20}$$

Thus we have

$$f_{X_1,X_2,X_3,X_4}(x_1, x_2, x_3, x_4) \propto \frac{1}{\sigma}\exp\left[-\frac{(x_1^2 + x_2^2 + x^3 + x^4 - \mu_0)^2}{2\sigma^2}\right], \tag{4.21}$$

In addition, the proposal distribution for the Metropolis-Hastings algorithm is defined to be the standard normal distribution for both dimensions. That is, for $x = (x_1, x_2, x_3, x_4)$, $y = (y_1, y_2, y_3, y_4)$ $\in \mathbb{R}^4$, $\mathcal{Q}(x, y) = \phi(y_1 - x_1)\phi(y_2 - x_2)\phi(y_3 - x_3)\phi(y_4 - x_4)$ where ϕ is the density function of the standard normal distribution. Then for any $x \in \mathbb{R}^4$, we have $\mathcal{N}(x) = \mathbb{R}^4$.

For reversible Partial Neighbor Sets, we must have reversibility for all $\mathcal{N}_i(x)$, which means $y \in \mathcal{N}_i(x) \iff x \in \mathcal{N}_i(y)$, $\forall x, y \in \mathcal{S}$. Thus, we consider $50 + 50$ neighbors for each reversible Partial Neighbor Set $\mathcal{N}_i(x)$, and we pick our neighbors by repeating the following steps for 50 times:

1. generates $\delta_1, \delta_2, \delta_3, \delta_4 \sim \text{Normal}(0, 1)$
2. for state $x = (x_1, x_2, x_3, x_4)$, put $y = (x_1+\delta_1, x_2+\delta_2, x_3+\delta_3, x_4+\delta_4)$ into the Partial Neighbor Set $\mathcal{N}_i(x)$;
3. to ensure the reversibility, also put $y' = (x_1 - \delta_1, x_2 - \delta_2, x_3 - \delta_3, x_4 - \delta_4)$ into the Partial Neighbor Set $\mathcal{N}_i(x)$;

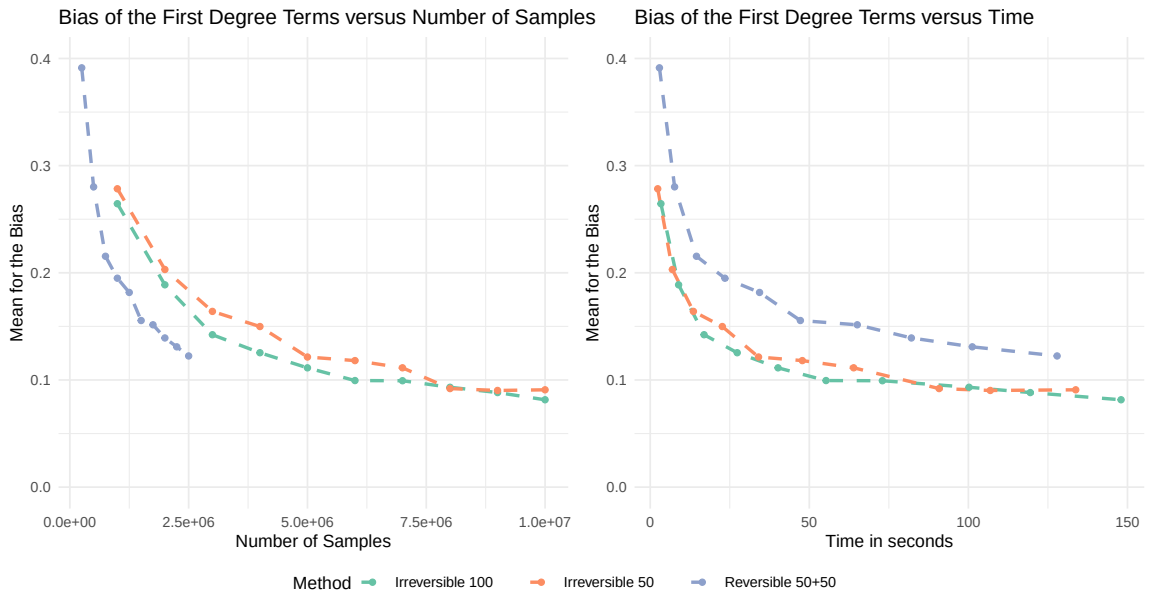

Figure 4.23: Sum of the Average Bias of X_i's between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for two methods: Metropolis algorithm and Unbiased PNS with the reversible and non-reversible partial neighbor set. We used the Donuts example with $\mu_0 = 9$ and $\sigma = 0.1$. Each dot within the plot represents the result of the average bias value and time used for 30 simulation runs given certain original sample sizes. $L_0 = 1000$.

After repeating the above steps 50 times, we can get a total of $50 + 50$ neighbors for the reversible Partial Neighbor Set $\mathcal{N}_i(x)$. For irreversible Partial Neighbor Sets, we don't need step 3. To make a fair comparison, we use two sizes of the Partial Neighbor Sets $|\mathcal{N}_i| = 50$ and 100.

The results are shown in Figure 4.23 and Figure 4.24. From the figures, we can see that the biases of the Markov chains by the irreversible Partial Neighbor Sets do decrease to near 0, which means such Markov chains do converge. In addition, it also performs better than the reversible ones. Does that mean we should use the irreversible Partial Neighbor Sets? Let's look at more simulation results.

In the previous simulation, we used a symmetric proposal distribution for the increment part of the states. That is, we generated $\delta_1, \delta_2, \delta_3, \delta_4 \sim \text{Normal}(0, 1)$, which is symmetric around 0. How about other proposal distribution? For example, $\text{Normal}(0.1, 1)$ can be a workable proposal distribution for the Metropolis-Hastings algorithm. Although $\text{Normal}(0.1, 1)$ is not symmetric around 0, the Markov chain by the Metropolis-Hastings algorithm will still converge to the target density. Will the irreversible PNS chain converge correctly?

The results are shown in Figure 4.25 and Figure 4.26. From the figures, we can see that the biases of the Markov chains by the irreversible Partial Neighbor Sets are not converging to 0, which means such Markov chains do not converge. This result illustrates that such an irreversible PNS chain will not always converge correctly. People need to be very careful when using irreversible PNS. We include this simulation because naturally, people prefer irreversible PNS to reversible ones since reversible ones include one more step of including the negative increment, and such a step

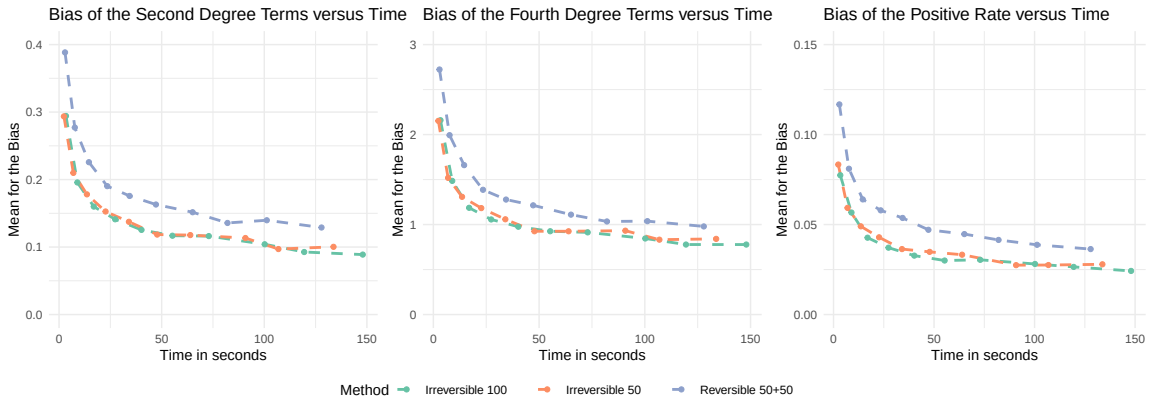

Figure 4.24: Sum of the average bias from the second-degree terms (left), the fourth-degree terms (middle), and the positive rate (right) between sampling and target density π as a function of average time in seconds for two methods: Metropolis algorithm and Unbiased PNS with the reversible and non-reversible partial neighbor set. We used the Donuts example with $\mu_0 = 9$ and $\sigma = 0.1$. Each dot within the plot represents the result of the average bias value and time used for 30 simulation runs given certain original sample sizes. $L_0 = 1000$.

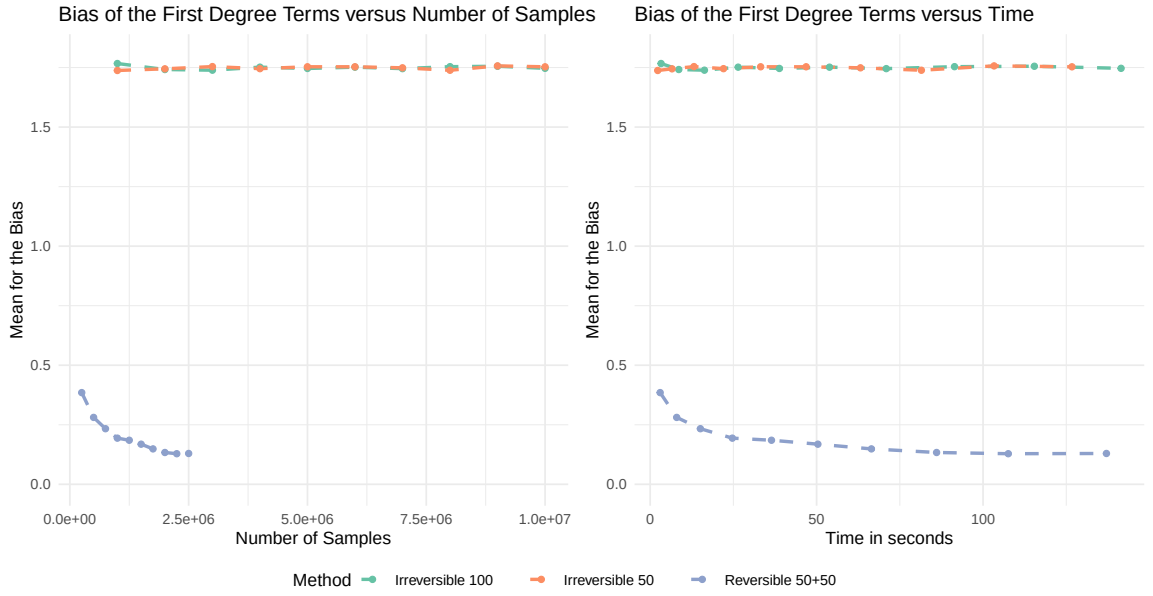

Figure 4.25: Sum of the Average Bias of X_i's between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for two methods: Metropolis algorithm and Unbiased PNS with the reversible and non-reversible partial neighbor set, where the proposal distribution is asymmetric with $\delta \sim Normal(0.1, 1)$. We used the Donuts example with $\mu_0 = 9$ and $\sigma = 0.1$. Each dot within the plot represents the result of the average bias value and time used for 30 simulation runs given certain original sample sizes. $L_0 = 1000$. According to the results here, we conclude that the non-reversible partial neighbor set is not always converging to the correct distribution.

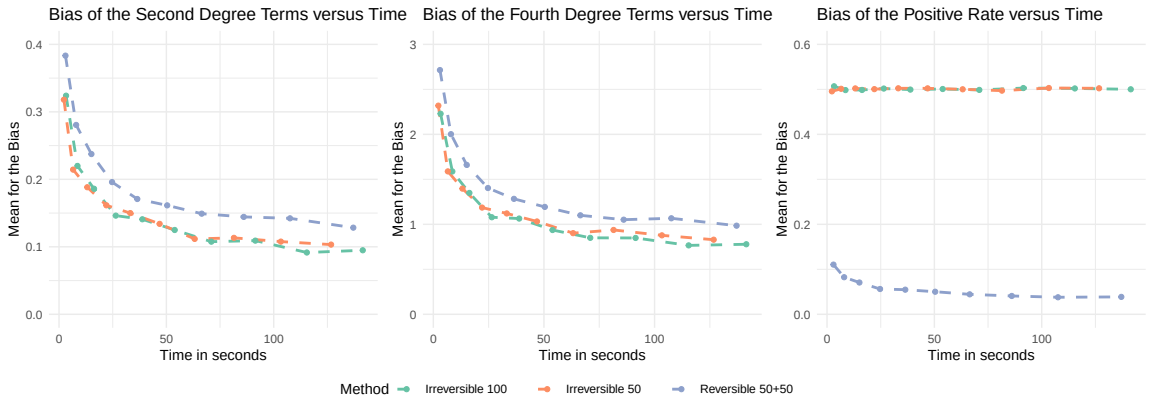

Figure 4.26: Asymmetric Proposal $\delta \sim Normal(0.1, 1)$: Sum of the average bias from the second-degree terms (left), the fourth-degree terms (middle), and the positive rate (right) between sampling and target density π as a function of average time in seconds for two methods: Metropolis algorithm and Unbiased PNS with the reversible and non-reversible partial neighbor set, where the proposal distribution is asymmetric with $\delta \sim Normal(0.1, 1)$. We used the Donuts example with $\mu_0 = 9$ and $\sigma = 0.1$. Each dot within the plot represents the result of the average bias value and time used for 30 simulation runs given certain original sample sizes. $L_0 = 1000$.

needs some time for the multicore hardware to communicate between cores. On the other hand, the first simulation shows that the irreversible PNS will converge to the target density under certain conditions, which can be another topic to explore in the future.

4.9 Combine PNS with other MCMC Techniques

4.9.1 Multiple Try Metropolis

In the Metropolis algorithm, we only propose one state at a time, and our Rejection-Free and PNS algorithms are adaptions of the Metropolis algorithm where we consider more neighbors at each step. Similarly, Multiple-Try Metropolis(MTM) is also an adaption of the Metropolis algorithm which proposed more than one neighbor to be tested at each step [56]. Some numerical studies show that MTM can be significantly better than the traditional Metropolis-Hastings algorithm [51].

The MTM algorithm is described by Algorithm 10. This algorithm is designed to increase the sampling speed by increasing the acceptance rate, although the step size is increased as well. Compare to our methods, MTM can only get one sample at a time by computing the $(2l-1)$ weights.

In addition, we can extend MTM to PNS as well. At each step, by applying the MTM transition rule for M times, we can obtain M proposals $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M$ with M corresponding acceptance probabilities $p_1, p_2, \ldots, p_M$. Then we can choose our next state by $\hat{P}(\mathbf{y}_i \mid X_{k-1}) \propto p_i$, and we can also calculate our multiplicity list by $m = 1 + G$ where $G \sim \text{Geometric}(p)$, and $p = \frac{1}{M}\sum_{i=1}^{M} p_i$.

However, we don't think it is a great idea to combine these two methods. Multiple-try Metropolis needs to compute the energy of $(2l-1)$ other states at every step. In addition, for each step of PNS, we have to do it for M times. Thus, we need to calculate $M \times (2l-1)$ functions for every step, and it can be overwhelming. In addition, the purpose of MTM is to increase the acceptance rate, while PNS works well for those cases with low acceptance rates. Thus, we think combining these

Algorithm 10 Multiple Try Metropolis

initialize X_0
choose a non-negative symmetric function $\lambda : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$
define $w(x, y) = \pi(x)\mathcal{Q}(x, y)\lambda(x, y)$, $\forall x, y \in \mathcal{S}$
for k in 1 to K **do**
  draw $y_1, y_2, \ldots, y_l$ independently from $Q(X_{k-1}, \cdot)$
  compute the weights $w(y_j, X_{k-1})$, $\forall j = 1, 2, \ldots, k$
  select $\mathbf{y}$ from the $y_1, y_2, \ldots, y_l$ based on $w(X_{k-1}, \cdot)$
  produce a reference set by drawing $z_1, \ldots, z_{l-1}$ from the distribution $Q(\mathbf{y}, \cdot)$
  set $z_l = X_{k-1}$
  accept $X_k = \mathbf{y}$ with probability $\min\left(1, \frac{w(y_1, X_{k-1}) + \cdots + w(y_l, X_{k-1})}{w(z_1, \mathbf{y}) + \cdots + w(z_l, \mathbf{y})}\right)$
end for

two methods won't be very efficient.

4.9.2 Barker's Rule

In Algorithm 1, we used the regular Metropolis-Hasting acceptance probability, which is

$$\min\left(1, \frac{\pi(Y)\mathcal{Q}(Y, X_{k-1})}{\pi(X_{k-1})\mathcal{Q}(X_{k-1}, Y)}\right).$$

On the other hand, we also have many other Markov chain kernels such as Barker's acceptance function [5], where we accept the proposal with probability

$$\frac{\pi(Y)\mathcal{Q}(Y, X_{k-1})}{\pi(Y)\mathcal{Q}(Y, X_{k-1}) + \pi(X_{k-1})\mathcal{Q}(X_{k-1}, Y)}.$$

We don't use these Markov chain kernels very often because based on Paskun-Tierney ordering [63], [80], these kernels are usually less efficient than the regular ones. However, under some special circumstances, we may need to use these kernels. Thus, we extend our PNS algorithm to these special Markov chain kernels as well.

Here use a simulation to show the change of the TVD values by sampling with PNS with Barker's acceptance function. Similar to the simulation from Section 4.3, we used the QUBO sampling example again. The only difference is that the non-zero elements were generated randomly by $Q_{i,j} \sim \text{Normal}(0, 1^2)$ instead of $\text{Normal}(0, 10^2)$, $\forall i \leq j$. We used a small variance here because we have to calculate $\log(\exp(y^T Q y) + \exp(x^T Q x))$, and with a large variance, calculating the exponential values may cause numerical overflows. The result is shown in Figure 4.27. The figure illustrates that the Markov chain with Barker's transition probabilities converges to the correct distribution, although the converging speed is slower than the regular Metropolis-Hastings kernel, which demonstrates that the Paskun-Tierney ordering works for PNS as well. Note that, the Markov chain produced by Metropolis-Hasting will converge correctly as long as the Markov chain kernels used is reversible. Similarly, we believe that PNS with any reversible Markov chain kernels works as well.

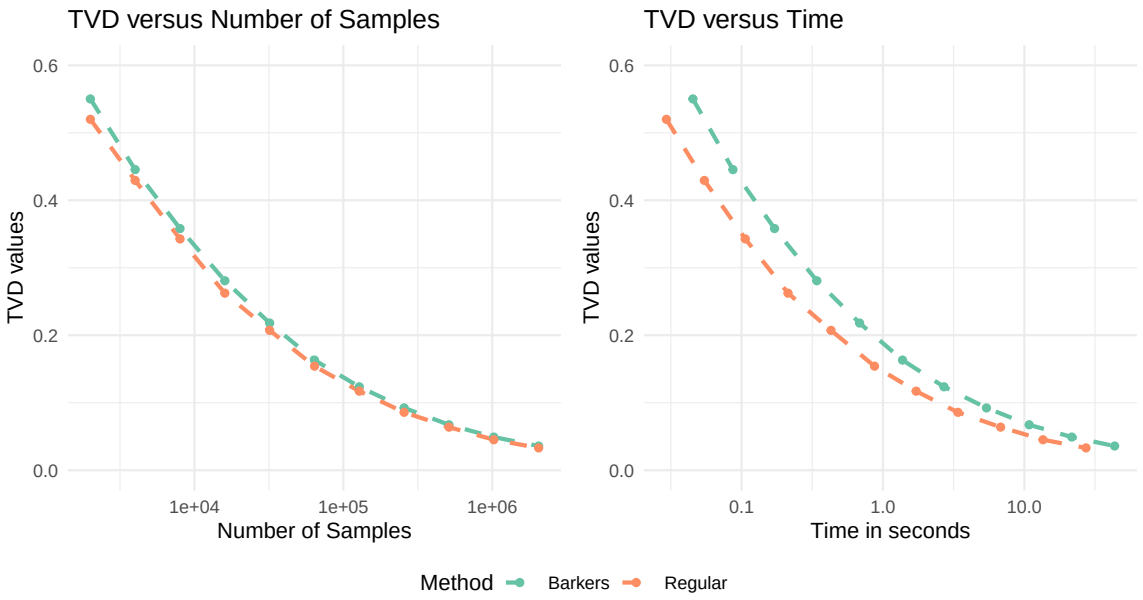

Figure 4.27: Average values of TVD between sampling and target density π as a function of the number of iterations (left) and average time in seconds (right) for two methods: Unbiased PNS with regular transition rule and Barker's transition rule. We used an upper triangular 16×16 QUBO matrix, generated randomly by $Q_{i,j} \sim N(0, 1^2)$ for upper triangular elements. Each dot within the plot represents the result of the average TVD value and time used for 100 simulation runs given certain original sample sizes. The original sample sizes for both algorithms are $\{2000, 4000, 8000, 16000, \ldots, 2048000\}$.We choose these sizes to get a close average CPU time for all three methods. For Unbiased PNS, we used $|\mathcal{N}_k| = 8$ and $L_0 = 100$.

Chapter 5

Optimization via Rejection-Free and Partial Neighbor Search

5.1 Optimization questions and Simulated Annealing Review

Optimization is the cornerstone of many areas. It plays a crucial role in finding feasible solutions to real-life problems, from mathematical programming to operations research, economics, management science, business, medicine, life science, and artificial intelligence [27]. Before the invention of linear and integer programming in the 1950s, optimization was characterized by several independent topics, such as optimum assignment, the shortest spanning tree, transportation, and the traveling salesman problem, which were then united into one framework [73]. Today, combinatorial optimization is essential in research because most problems originate from practice and are dealt with daily [73]. Optimization questions aim to find an optimal solution to maximize or minimize a real function within a given state space. Sometimes, a feasible solution with the corresponding function value near the optimal solution is also acceptable. The process of finding an optimal or feasible solution to some complex combinatorial optimization problems may take a considerable amount of time. In particular, no algorithm for NP-hard problems can guarantee that the optimal state of the problem will be found within a limitation governed by a polynomial based on the input length [29].

Among all complex optimization problem solvers, metaheuristics are usually nature-inspired [8]. They are designed to select a heuristic that often arrives at a feasible solution instead of an optimal one. The Simulated Annealing algorithm [47], based on the Metropolis steps [59] at decreasing temperatures, is a typical method of this kind, as we now review.

Simulated Annealing, as introduced by [47], is widely used to solve combinatorial optimization problems, such as approximating the optimal values of functions with many variables [70]. Although there is some theory to prove that Simulated Annealing will converge to the optimal solution almost surely with sufficiently slow cooling schedules [62], for many complex optimization problems, such as NP-hard problems, there is no guarantee that this algorithm will provide an optimal solution within a reasonable amount of time. On the other hand, Simulated Annealing can give reasonable, feasible solutions quickly [1]. Discrete Simulated Annealing contains the following essential elements [7]:

1. A state space $\mathcal{S}$.

2. A real-valued target distribution π on $\mathcal{S}$. The ultimate goal for the Simulated Annealing is to find $Y \in \mathcal{S}$ such that $\pi(Y) \geq \pi(X)$, $\forall X \in \mathcal{S}$. However, for many circumstances, a good feasible solution is acceptable.

3. $\forall X \in S$, $\exists$ a proposal distribution $\mathcal{Q}(X, \cdot)$ where $\int_{Y \in \mathcal{S} \backslash \{X\}} \mathcal{Q}(X, Y) = 1$.

4. $\forall X \in \mathcal{S}$, $\exists\, \mathcal{N}(X) = \{Y \in \mathcal{S} \mid \mathcal{Q}(X, Y) > 0\} \subset \mathcal{S} \backslash \{X\}$, called the neighbors of X.

5. A non-increasing function $T : \mathbb{N} \to (0, \infty)$, called the Cooling Schedule. $T(k)$ is called the temperature at step $k \in \mathbb{N}$.

6. An initial State $X_0 \in \mathcal{S}$.

With the above elements, the Simulated Annealing algorithm, which consists of a discrete time-inhomogeneous Markov Chain $\{X_k\}_{k=0}^{K}$ can be generated by Algorithm 11. Algorithm 11 is designed to converge to states X_k with nearly-maximal values of $\pi(X_k)$, though that is not guaranteed. Note that the algorithm can also be formulated using log values for better numerical stability.

Algorithm 11 Simulated Annealing

initialize X_0
for k in 1 to K **do**
  random $Y \in \mathcal{N}(X_{k-1})$ based on $\mathcal{Q}(X_{k-1}, \cdot)$
  random $U_k \sim \text{Uniform}(0, 1)$
  if $U_k < \left[\frac{\pi(Y)}{\pi(X_{k-1})}\right]^{1/T(k)}$ **then**
    ▷ accept with probability $\min\left\{1, \left[\frac{\pi(Y_k)}{\pi(X_{k-1})}\right]^{1/T(k)}\right\}$
    $X_k = Y$ ▷ accept and move to state Y
  else
    $X_k = X_{k-1}$ ▷ reject and stay at X_{k-1}
  end if
end for

The Simulated Annealing algorithm, just like the Metropolis-Hastings algorithm, may be inefficient with respect to rejections. In order to improve the performance of Simulated Annealing, we adopt the Rejection-Free algorithm for sampling from Chapter 3 into an optimization version. Additionally, Rejection-Free may experience inefficiency when it enters local extreme areas. Therefore, we can also adopt our Partial Neighbor Search (PNS) for sampling from Chapter 4 to optimization questions as well to further enhance its efficiency.

5.2 Rejection-Free algorithm for optimization

In addition to sampling, the Rejection-Free algorithm can also be applied to optimization problems. Given a set $\mathcal{S}$ and a real-valued target distribution π on the set $\mathcal{S}$, we can use the Rejection-Free algorithm to find $X \in \mathcal{S}$ that maximizes $\pi(X)$ by Algorithm 12. Algorithm 12 is again designed to converge to states X_k with nearly-maximal values of $\pi(X_k)$, with greater efficiency by avoiding rejections, though that is again not guaranteed. Although the purpose of sampling and optimization are different, regarding the implementation, Rejection-Free for optimization is only different from Rejection-Free for sampling by getting rid of the multiplicity list $\{M_k\}$.

Algorithm 12 Rejection-Free for Optimization (Discrete Cases)

initialize J_0
for k in 1 to K **do**
  for $Y \in \mathcal{N}(J_{k-1})$ **do**
    ▷ only works for finite neighbors
    calculate $q(Y) = \mathcal{Q}(Y, J_{k-1}) \min\{1, [\frac{\pi(Y)}{\pi(J_k)}]^{\frac{1}{T(k)}}\}$
    ▷ the transition prob. from J_{k-1} to Y
  end for
  choose $J_k \in \mathcal{N}(J_{k-1})$ such that $\hat{P}(J_k = Y \mid J_{k-1}) \propto q(Y)$
  ▷ choose the next jump chain State
end for

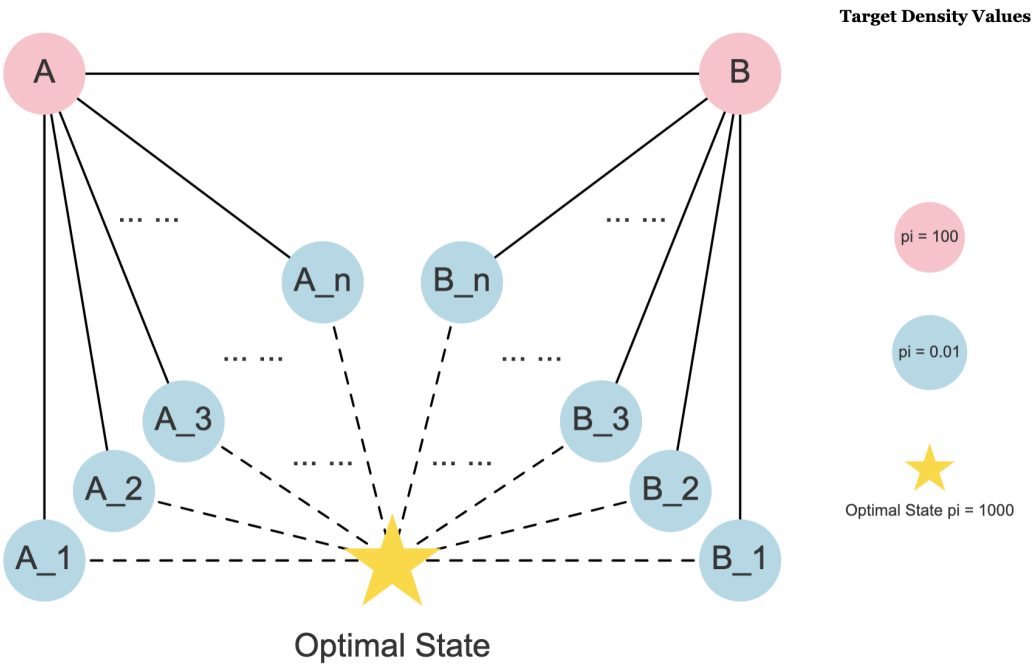

Figure 5.1: Illustration of the local maximum area in an optimization problem where both Simulated Annealing and Rejection-Free may get stuck. The target distribution π has the following function values: $\pi(A) = \pi(B) = 100$, $\pi(A_1) = \pi(A_2) = \cdots = \pi(A_n) = \pi(B_1) = \pi(B_2) = \cdots = \pi(B_n) = 0.01$.

Although the Rejection-Free algorithm for optimization can help reduce the inefficiency of rejections, local maximum areas of π can still be a problem. For example, we want to find $X \in S$, which maximizes $\pi(X)$ from a state space starting at state A in Figure 5.1. Here, we use a uniform proposal distribution $\mathcal{Q}$ on the neighbor sets $\mathcal{N}$ as shown in Figure 5.1. We will have many rejections if we constantly use Simulated Annealing with $T \equiv 1$. Note that, $\pi(A_1) = \pi(A_2) = \cdots = \pi(A_n) = 0.01$ while $\pi(A) = \pi(B) = 100$. The probability of escaping from A is $\frac{1}{n+1} + \frac{n}{n+1} \times \frac{1}{10000}$, where $\frac{1}{n+1}$ represents the probability of moving from state A to state B, and $\frac{n}{n+1} \times \frac{1}{10000}$ is the probability of moving from state A to $A_1, A_2, \ldots, A_n$. Cooling Schedules can help reduce the probability of rejection at the beginning of Simulated Annealing since T should be large at the beginning. However, as we move on in Simulated Annealing, we will be more and more likely to be trapped by local maximum areas like this. The Rejection-Free algorithm for optimization can produce some speedup in this case, but the Rejection-Free chain will still be stuck by the local maximum area $\{\pi(A), \pi(B)\}$. If n, the number of other neighbors for A and B, is small, this chain will be switching between A and B for a really long time, since

$$\begin{aligned} &\hat{\mathbf{P}}(J_1 = B \mid J_0 = A) = \frac{\min\{1, \frac{\pi(B)}{\pi(A)}\}}{\sum_{z \neq A} \min\{1, \frac{\pi(z)}{\pi(A)}\}} = \frac{1}{1 + 0.0001 \times n} \approx 1 \\ &\hat{\mathbf{P}}(J_1 = A \mid J_0 = B) \approx 1. \end{aligned} \tag{5.1}$$

To help our Markov chain escape from those local maximums in optimization, we propose another method called Partial Neighbor Search based on the Rejection-Free algorithm.

5.3 Partial Neighbor Search for Optimization

Partial Neighbor Search (PNS) is an algorithm based on the Rejection-Free, also designed as a Markov chain used for optimization as described in Algorithm 13. Algorithm 13 is again designed to converge to states X_k with nearly-maximal values of $\pi(X_k)$, with greater efficiency by avoiding both rejections and traps in local maximum areas.

Algorithm 13 Optimization Partial Neighbor Search

initialize J_0
for k in 1 to K **do**
  pick $\mathcal{N}_k(J_{k-1}) \subset \mathcal{N}(J_{k-1})$ $(\star)$
  for $Y \in \mathcal{N}_k(J_{k-1})$ **do** ▷ Only neighbors in $\mathcal{N}_k$ will be considered
    calculate $q(Y) = \mathcal{Q}(Y, J_{k-1}) \min\{1, [\frac{\pi(Y)}{\pi(J_k)}]^{\frac{1}{T(k)}}\}$
    ▷ the transition prob. from J_{k-1} to Y
  end for
  choose $J_k \in \mathcal{N}_k(J_{k-1})$ such that $\hat{\mathbf{P}}(J_k = Y \mid J_{k-1}) \propto q(Y)$
  ▷ choose the next Jump Chain State
end for

The $(\star)$ step in Algorithm 13 is the key of PNS. At this step, $\mathcal{N}_k(J_{k-1})$ could be random 50% of the elements from $\mathcal{N}(J_{k-1})$. In Section 5.6, we will explore many other choices for the $(\star)$ step to figure out the best strategy. Moreover, for continuous cases, PNS can be applied, and we only need to ensure the Partial Neighbor Sets $\mathcal{N}_k$ are always finite, $\forall k$. On the other hand, Algorithms 4 and Algorithm 12 for Rejection-Free only work for discrete cases where the number of neighbors for all

states must be finite, and we will illustrate these by an optimization example in continuous cases in Section 5.10.

The motivation for PNS is simple: we have a better chance of escaping from the local maximum area if we force the algorithm to avoid some neighbors randomly. For example, in Figure 5.1, if we only consider half of the neighbors at state A, then we may disregard state B with probability 50%, then we have a probability of at least 50% of selecting a state from $\{A_1, A_2, \ldots, A_n\}$ as our next state in the PNS chain. If this occurs, we are more likely to escape from the local maximum area $\{\pi(A), \pi(B)\}$.

5.4 Application to the QUBO question

Here we use the QUBO question as an example again. For the optimization version as it supposes to be, the QUBO questions aims to find the optimal or workable solution for the function $x^T Qx$. It is known to be NP-hard [37], so it is common to use Simulated Annealing to find the optimal or workable solution. This problem can now be addressed using our PNS algorithm.

For a given N by N matrix Q (usually upper triangular), the QUBO question aims to find

$$\arg\max x^T Qx, \text{ where } x \in \{0,1\}^N \tag{5.2}$$

(Sometimes arg min is used in place of arg max, which is equivalent to taking the negative of Q, so for simplicity, we focus on the arg max version here.)

As part of our algorithm, we use a uniform proposal distribution among all neighbors where the neighbors are defined as binary vectors with Hamming distance 1. That is, $\mathcal{Q}(X,Y) = \frac{1}{N}$ for $\forall Y \in \mathcal{N}(X)$, where $Y \in \mathcal{N}(X) \iff |X - Y| = \sum_{i=1}^{N} |X_i - Y_i| = 1$, $\forall X, Y \in \{0,1\}^N$. We randomly choose half of the neighbors at each step of PNS, which means we only consider a random subset $\mathcal{N}_K(x) \subset \mathcal{N}(x)$ whose cardinality is $|\mathcal{N}_k(X)| = \frac{1}{2}|\mathcal{N}(X)| = \frac{1}{2}N$ for $\forall X \in \{0,1\}^N$. In addition, the target distribution $\pi(x) = \exp\{x^T Qx\}$, since we need the target distribution to be positive all time to use the Cooling Schedule, and maximizing $x^T Qx$ is the same as maximizing $\exp\{x^T Qx\}$. Furthermore, $T(k)$ represents the temperature at step k for the cooling schedule here.

We compare Simulated Annealing, Rejection-Free for Optimization, and PNS in 1000 simulation runs. We randomly generate a 200 by 200 upper triangular as the QUBO matrix Q. The non-zero elements from Q were generated randomly by $Q_{i,j} \sim \text{Normal}(0, 100^2)$, $\forall i \leq j$.

The result for the simulation is shown in Figure 5.2. Here, we used a violin plot to summarize the results. The violin plot uses the information available from local density estimates and the basic summary statistics to provide a useful tool for data analysis and exploration [42]. The violin plot combines two density traces on both sides and three quantile lines (25%, 50%, and 75%) to reveal the data structure. In addition, we added a long segment of the bottom layer as the mean for the values. We also added a short segment on the y-axis to help compare the mean values.

From Figure 5.2, we can see that the PNS is always the best in all four different cooling schedules. Note that the number of iterations used for Simulated Annealing is $200,000$ for Simulated Annealing while they are 1000 for both Rejection-Free and PNS. We used these many iterations because we need to consider 200 neighbors at each iteration in Rejection-Free, while we only need to consider one neighbor for each iteration in Simulated Annealing. If we proceed with all three algorithms on

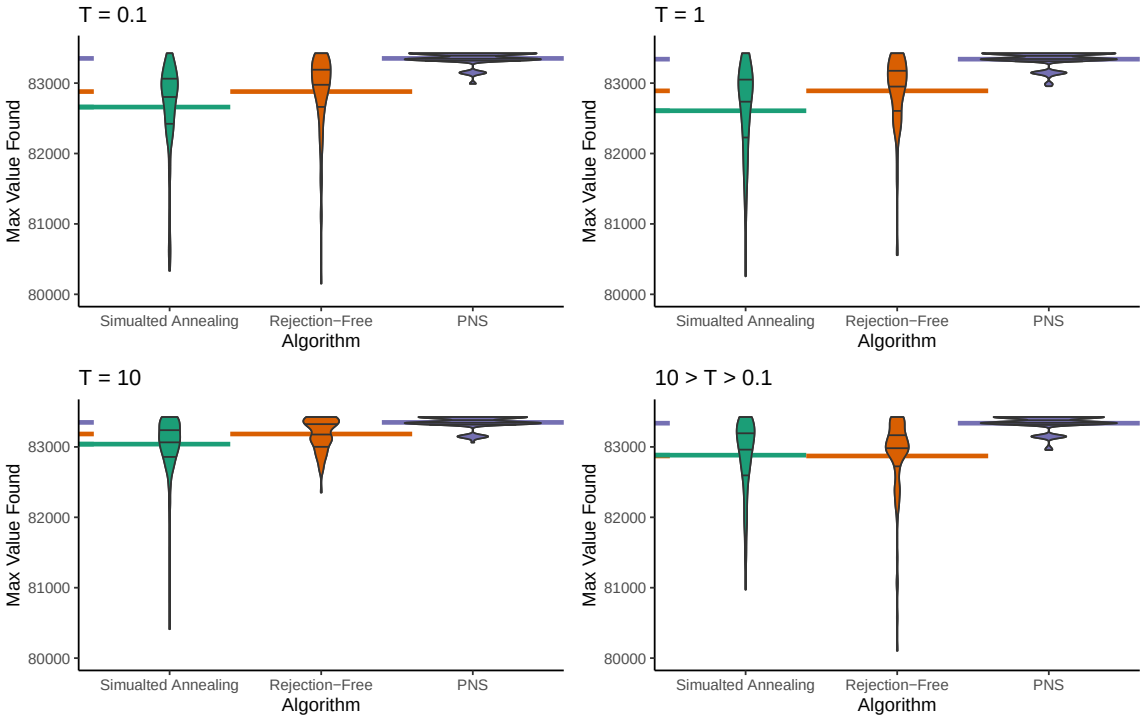

Figure 5.2: Comparison of Simulated Annealing, Rejection-Free, and PNS in terms of the highest (log) target distribution value $\log \pi(x) = x^T Q x$ being found, for a random upper triangular QUBO matrix Q where the non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules where $T(k) = 0.1$, 1 and 10 constantly, and $T(k)$ being geometric from 10 to 0.1 are used here. The number of iterations for Simulated Annealing is 200,000, and the numbers of iterations for Rejection-Free and PNS are 1000. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

a single-core machine, the run time of a single simulation run for simulated Annealing is about 20 seconds; the run time for Rejection-Free is about 10 seconds; the run time for PNS is only 5 seconds. In addition, parallel hardware can increase the speed of both Rejection-Free and PNS by distributing the calculation of the transition probabilities for different neighbors onto different cores. Besides that, we can also use multiple replicas at different temperatures, such as in parallel tempering, or deploy a population of replicas at the same temperature [76]. Combining these methods by parallel computing can yield 100x to 10,000x speedups for both Rejection-Free and PNS [76].

In the above example, the improvement in the efficiency of Rejection-Free is not hard to understand. The performance of PNS is somehow counter-intuitive. Compared to Rejection-Free, why would we get a better result by considering fewer neighbors at each step? To illustrate how PNS works, we can look closely at the Markov chains generated in the above example.

5.5 Understanding the improvement of PNS

In this section, we found a local maximum area for the target distribution π purposefully in the previous QUBO example in Section 5.4 by looking at the final results from the simulation runs from the previous section. Many Rejection-Free chains stopped at this local maximum area after 1000 iterations. For this local maximum area, the target distribution value is around 82600, and this local maximum area contains three states whose target distribution values are much larger than all their other neighbors. Thus, this local maximum can trap the regular Rejection-Free chain for a long time, just like the cases we mentioned in Figure 5.1. We can plot the Markov chains by PNS with the target distribution values for all the neighbors by Rejection-Free and the random subset of neighbors by PNS in the form of boxplots. The boxplot of the first 30 steps from the first simulation in PNS is shown in the first plot in Figure 5.3.

From the first plot in Figure 5.3, most of the target distribution values within the boxplot are not useful since they are too small to be picked by the algorithm. Therefore, we only need to consider the important neighbors likely to be chosen. Firstly, for each state J_k in the Markov Chain, we find the max value among all the transition probabilities, and we define the important neighbors to be those neighbors whose transition probability is larger than $\exp\{-10\}$ times the highest transition probability among all neighbors. That is, for each J_k from the chain, we find $q(Y_0) = \max\{q(Y) \mid Y \in \mathcal{N}(J_k)\}$, and then we define $\{Y \mid Y \in \mathcal{N}(J_k), q(Y) > \exp\{-10\} \times q(Y_0)\}$ to be important neighbors for J_k. This time, we only have several important neighbors at each step. Thus, we used points instead of boxplots to show the important neighbors. The result from Rejection-Free and PNS is also shown in Figure 5.3.

From the second plot in Figure 5.3, the red dots represent the important neighbors, and the pink line means the Rejection-Free chain. We can see that this local maximum area of three states can easily trap the Rejection-Free chains because their target distribution values are much higher than all other neighbors. Thus, the important neighbors for any of these three states are only the remaining two, and the Rejection-Free chain will be switching between these three for a long time. At the same time, the blue dots in the second plot represent the important neighbors if we start to do PNS from that state. Although we did not apply PNS in the second plot, we still put the random subset for PNS there as a comparison. From the blue dots in the second plot, we can say that if we perform PNS, then the Markov chain can escape from this local maximum area faster since some

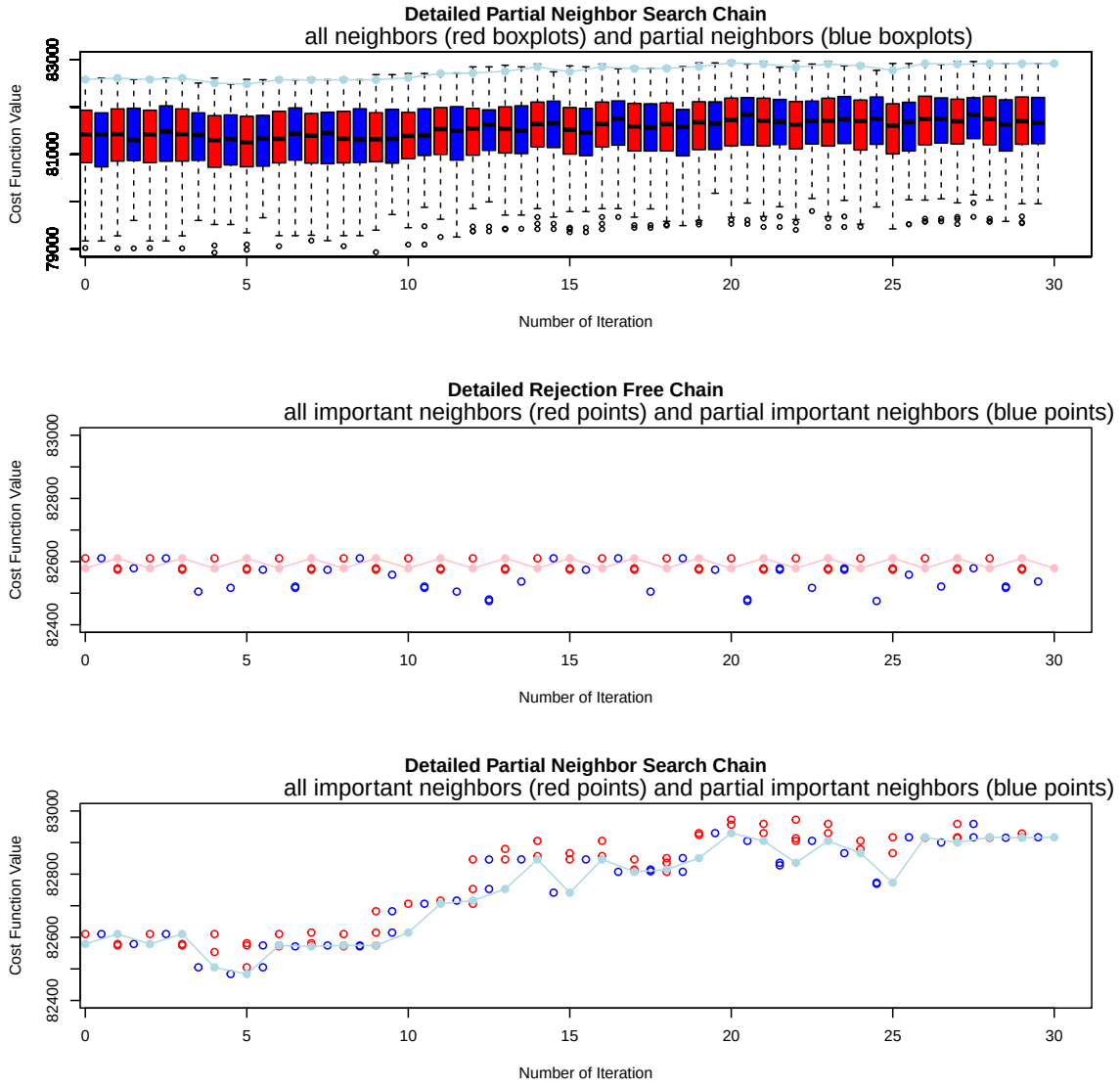

Figure 5.3: The detailed Markov Chains from Rejection-Free (the pink chain in the second plot) and PNS (the light blue chain in the first and the third plot). The red box plots in the first plot represent the target distribution values for all neighbors, and the blue box plots represent the partial neighbors. Most of these values are useless because they are too small to be picked by the Markov chain. The second and the third plots only show the important neighbors, defined as those whose transition probability is larger than $\exp(-10)$ times the highest transition probability among all neighbors. Here, red points represent all important neighbors, and blue points mean important neighbors of a random subset of all neighbors used for PNS. The Rejection-Free Chain switches between three local maximum states all the time while the PNS chain escapes from the local maximum area after five iterations.

groups of the blue dots do not contain any of these three states with high target distribution values.

On the other hand, the third plot in Figure 5.3 shows that the PNS chain (blue line) escapes from this local maximum area within five steps. Again, the blue dots represent the important neighbor from PNS, and the red dots represent the important neighbor if we start to perform Rejection-Free from that step. For each step of PNS within the local maximum area of three states, the Markov Chain has the probability of 25% to include neither of the remaining neighbors from the three states. Thus, PNS helped the Markov chain to escape from this local maximum area. In addition, in the middle part of the PNS chain, when the target distribution value of the PNS chain is increasing, we usually have more than one important neighbor. For example, if we have three important neighbors, we only have 12.5% for considering none of them by PNS.

Thus, the PNS is better than Rejection-Free because the PNS performs much better than the Rejection-Free algorithm when the local maximum areas trap the Markov chain. On the other hand, PNS is not much worse than Rejection-Free when the Markov chain is increasing with respect to the target distribution value.

This section uses 50% random partial neighbors for each step. We have many other choices, and we will consider and compare these choices in the next section.

5.6 Optimal subset choice for Partial Neighbor Search

We formally defined the Partial Neighbor Sets when we prove the convergence theorem for the sampling version of PNS in Section 4.5. For optimization, the choice of Partial Neighbor Sets can be chosen not that strictly. We simply need to make sure, with the current choice of Partial Neighbor Sets, the Markov chain is able to find the global optimal solution. However, in practice, when we don't know too much about the model, choosing Partial Neighbor Sets satisfying all the conditions in Definition 4.5.2 is the best strategy.

Now, we compare the four different ways to choose the proposal distribution $\{\mathcal{Q}_k, \mathcal{N}_k\}$ for PNS in the $(\star)$ step in Algorithm 13:

- Method A (random subset every step): The Partial Neighbor Sets $\mathcal{N}_k$ are randomized for every step, where $|\mathcal{N}_k(X)| = \frac{1}{2} \times |\mathcal{N}(X)|$. $\mathcal{Q}_k$'s are defined accordingly.
- Method B (random subset every 10 steps): The Partial Neighbor Sets $\mathcal{N}_k$ are randomized for once 10 steps, where $|\mathcal{N}_k(X)| = \frac{1}{2} \times |\mathcal{N}(X)|$. That is, $\mathcal{N}_{10\times k+1} = \mathcal{N}_{10\times k+2} = \cdots = \mathcal{N}_{10\times k+10}$ for $\forall k \in \mathbb{N}$. $\mathcal{Q}_k$'s are defined accordingly.
- Method C (systematic subset every step): Before we start our Markov Chain, we define two Partial Neighbor Sets $\mathcal{N}_1$ and $\mathcal{N}_2$, where $|\mathcal{N}_1(X)| = |\mathcal{N}_2(X)| = \frac{1}{2} \times |\mathcal{N}(X)|, \mathcal{N}_1(X) \cap \mathcal{N}_2(X) = \emptyset$. For step k of the Markov chain, we only randomly generate $r_k \in \{1, 2\}$, and apply $\mathcal{N}_{r_k}$ for step k. $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are defined accordingly.
- Method D (systematic subset every 10 steps): Before we start our Markov Chain, we define two Partial Neighbor Sets $\mathcal{N}_1$ and $\mathcal{N}_2$, where $|\mathcal{N}_1(X)| = |\mathcal{N}_2(X)| = \frac{1}{2} \times |\mathcal{N}(X)|, \mathcal{N}_1(X) \cap \mathcal{N}_2(X) = \emptyset$. For every ten steps of the Markov chain, we only randomly generate $r_k \in \{1, 2\}$ and apply $\mathcal{N}_{r_k}$. That is $r_{10\times k+1} = r_{10\times k+2} = \ldots = r_{10\times k+10}$ for $\forall k \in \mathbb{N}$. $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are defined accordingly.

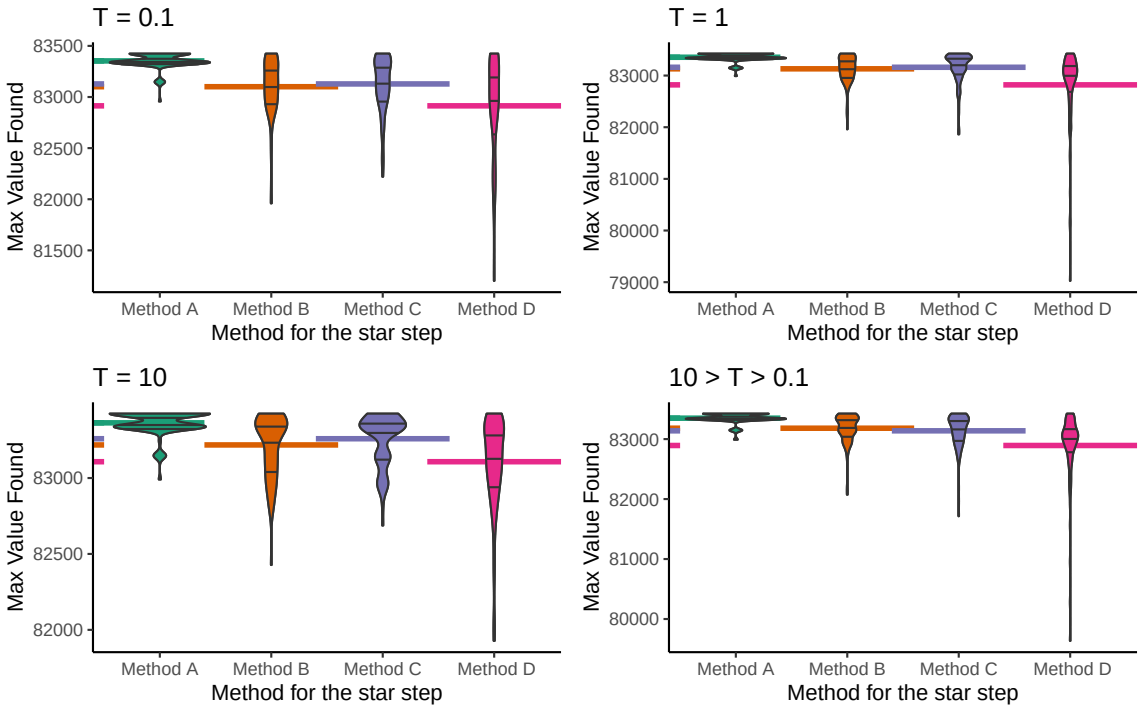

Figure 5.4: Comparison of different methods to choose the subsets for PNS, in terms of the highest (log) target density value $\log \pi(x) = x^T Q x$ found. Method A: random subset every step; method B: random subset every ten steps; method C: systematic subset every step; method D: systematic subset every ten steps. Random upper triangular QUBO matrix where the non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules where T = 0.1, 1, and 10 for all n, and T being geometric from 10 to 0.1, are used here. The number of iterations for all methods is 1000. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

Again, we use the 200×200 QUBO example. The settings for the simulation are the same as in Section 5.4. For Method C and D, the two Partial Neighbor Sets $\mathcal{N}_1$ and $\mathcal{N}_2$ are defined to be flipping the first 100 entries in x and flipping the last 100 entries in x. The result for the simulation is shown in Figure 5.4. This figure shows that the random subset at every step (Method A) performs the best in all four Cooling Schedules. Therefore, we will keep using Method A in all later parts.

In addition, we used to choose Partial Neighbor Sets with half elements from all neighbors. Now we compare the Partial Neighbor Sets with cardinality of $|\mathcal{N}(X)| \times \{1, \frac{3}{4}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}\}$ by the same simulation settings as before. From Figure 5.5, we can see that $\frac{1}{3}, \frac{1}{4}, \frac{1}{5}$ are overall the best among all the choices. Thus, we can conclude that Partial Neighbor Sets with around 25% of the neighbors being considered at each step are the best for the QUBO question stated above.

Therefore, we conclude that our best method to do optimization for the 200×200 QUBO question is Algorithm 14.

5.7 Comparison with Tabu Rejection-Free algorithm

Tabu search [35] [36] is also a methodology in optimization that guides a local heuristic search procedure to explore the solution space beyond local optimality. The idea of Tabu search is to

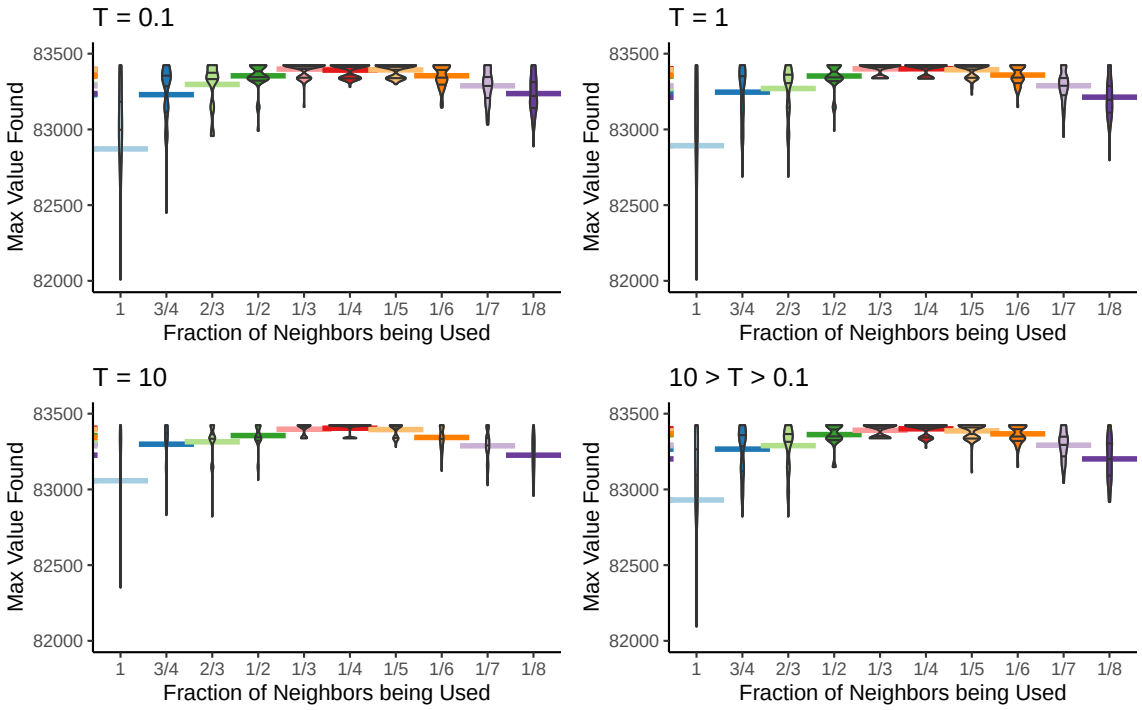

Figure 5.5: Comparison of different sizes of the random subsets for PNS, in terms of the highest (log) target density value $\log \pi(x) = x^T Q x$ being found. Subset sizes are $N \times \{1, \frac{3}{4}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \frac{1}{7}, \frac{1}{8}\}$. Random upper triangular QUBO matrix where the non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules where T = 0.1, 1, and 10 for all n, and T being geometric from 10 to 0.1, are used here. The number of iterations for all methods is 1000. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

Algorithm 14 Partial Neighbor Search for the 200 by 200 QUBO question

initialize J_0
for k in 1 to K **do**
  randomly pick $\mathcal{N}_k(J_{k-1}) \subset \mathcal{N}(J_{k-1})$ where $|\mathcal{N}_k(J_{k-1})| = 50$
    ▷ Only 50 out of the 200 neighbors will be considered
  for $Y \in \mathcal{N}_k(J_{k-1})$ **do**
    calculate $q(Y) = \min\{1, [\frac{\exp(Y^T Q Y)}{\exp(J_{k-1}^T Q J_{k-1})}]^{\frac{1}{T(k)}}\}$
      ▷ the transition prob. from J_{k-1} to Y
  end for
  choose $J_k \in \mathcal{N}_k(J_{k-1})$ such that $\hat{\mathbf{P}}(J_k = Y \mid J_{k-1}) \propto q(Y)$
    ▷ choose the next Jump Chain State
end for

prohibit access to specific previously-visited solutions. Tabu search is the most intuitive method to help the Markov Chain escape from local maximum areas, as in Figure 5.1. After moving from state A to state B, we must choose our next state among $\{B_1, B_2, \ldots, B_N\}$. We can combine our Rejection-Free algorithm for optimization with Tabu search and then compare this new method to the PNS by the QUBO question. Note that we do not need to record all visited states since we are almost impossible to revisit a state after a certain number of steps. Thus, we only need to record the last several steps and prohibit our Markov chain from revisiting them. The new algorithm is formulated as Algorithm 15.

Algorithm 15 L steps Simplified Tabu Rejection-Free for optimization

initialize J_0
for k in 1 to K **do**
  for $Y \in \mathcal{N}(J_{k-1}) \backslash \{J_{k-2}, \ldots, J_{k-L-1}\}$ **do**
    ▷ Remove states from the last L steps
    $q(Y) = \min\{1, [\frac{\exp(Y^T Q Y)}{\exp(J_k^T Q J_k)}]^{\frac{1}{T(k)}}\}$
    ▷ the transition prob. from J_{k-1} to Y
  end for
  choose $J_k \in \mathcal{N}_k(J_{k-1})$ such that $\hat{\mathbf{P}}(J_k = Y \mid J_{k-1}) \propto q(Y)$
    ▷ choose the next Jump Chain State
end for

Here, we compare PNS with L-step Simplified Tabu Rejection-Free for $L = 1, 2, 3, \ldots, 9$. Again, we randomly generate a 200 by 200 upper triangular QUBO matrix. The non-zero elements from the 200 by 200 upper triangular matrix Q were generated randomly with $Q_{i,j} \sim N(0, 100^2)$ for $i < j$. Note that we need to consider about 200 neighbors at each step for both Rejection-Free and Simplified Tabu Rejection-Free, while we only need to consider 50 neighbors at each iteration for PNS. If we proceed with the algorithms with a single-core implementation, Rejection-Free and Tabu Rejection-Free need about four times longer than PNS with the same number of steps. Therefore, we can compare the PNS with $4 \times 100 = 400$ iterations with the other methods to get a fair comparison for the program on a single core. Note that we are using this many numbers of steps here because 400 steps are enough for PNS to find a good enough answer. The result for the simulation is shown in Figure 5.6. From this plot, we can see that PNS performs much better than Rejection-Free and Simplified Tabu Rejection-Free.

5.8 Application to Knapsack problem

The Knapsack problem is another well-known NP-hard problem in optimization [72]. We consider the simplest 0-1 Knapsack problem here. Given a knapsack of max capacity W and N items with corresponding values $\{v_i\}_{i=1}^N$ and weights $\{w_i\}_{i=1}^N$, we want to find a finite number of items among all N items which can maximize the total value while not exceeding the max capacity of the knapsack. That is, for given $W > 0$, $\{v_i\}_{i=1}^N > 0$ and $\{w_i\}_{i=1}^N > 0$, find a sequence of N binary variable $\{X_i\}_{i=1}^N \in \{0, 1\}$ to maximize

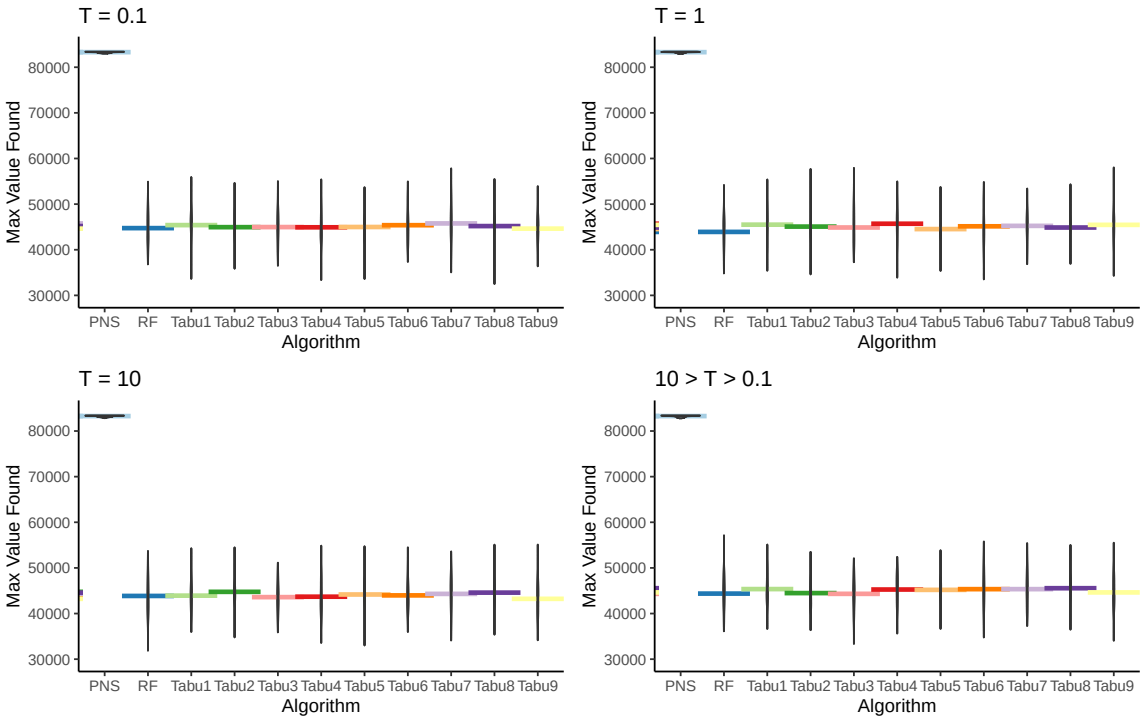

Figure 5.6: Comparison of PNS, Rejection-Free, and 1-Step to 9-steps Simplified Tabu Rejection-Free, in terms of the highest (log) target density value $\log \pi(x) = x^T Q x$ found. Random upper triangular QUBO matrix where the non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules where $T = 0.1$, 1, and 10 constantly, and T being geometric from 10 to 0.1, are used here. The run time for all algorithms on a single-core implementation is about the same. The number of iterations for PNS is 400, and the number of iterations for all other methods is 100. The colored segments represent the mean values.

$$\begin{aligned} &\sum_{i=1}^{N} v_i X_i \\ \text{subject to } &\sum_{i=1}^{N} w_i X_i \leq W \end{aligned} \tag{5.3}$$

Since the Knapsack problem is NP-hard, we can use the Simulated Annealing algorithm to find a feasible solution. For this simulation, we set $W = 100,000$. We randomly generate $N = 1000$ items where the values and weights are random by $w_i, v_i \sim \text{Poisson}(1000)$. The mean and the variance for Poisson(1000) are both 1000. Suppose we want to find a binary vector $X = (X_1, X_2, \ldots, X_N)^T$ of dimension N to maximize $v^T X$ subject to $w^T X \leq W$.

Again, we used a uniform proposal distribution among all neighbors where the neighbors are defined as binary vectors with Hamming distance 1. That is, $\mathcal{Q}(X, Y) = \frac{1}{N}$ for $\forall Y \in \mathcal{N}(X)$, where $Y \in \mathcal{N}(X) \iff |X - Y| = \sum_{i=1}^{N} |X_i - Y_i| = 1$, $\forall X, Y \in \{0,1\}^N$. We randomly choose half of the neighbors at each step for PNS. That is, $|\mathcal{N}_k(X)| = \frac{1}{2}|\mathcal{N}(X)| = 500$ for $\forall X \in \{0,1\}^N$. Moreover, the target density $\pi(X) = \mathbb{1}(w^T X \leq W) \times v^T X$, where $\mathbb{1}$ represents the indicator function. In addition, $T(k)$ represents the temperature at step k for the Cooling Schedule here.

Again, we compare Simulated Annealing, Rejection-Free, and PNS here. The result is shown in Figure 5.7. The plot shows that Rejection-Free for optimization and PNS algorithm are better than the regular Simulated Annealing algorithm in all four Cooling Schedules. Again, for the simulation shown in Figure 5.7, the numbers of iterations used for the three methods are set to be different to have a fair comparison between three methods. We set the number of iterations for Simulated Annealing to be $1,000,000$. The numbers of iterations for Rejection-Free and PNS are 1000 since we need to consider 1000 neighbors at each iteration for Rejection-Free for optimization. In contrast, we only need to consider one neighbor for each iteration in Simulated Annealing.

This result shows that PNS is not always that much better than Rejection-Free when the number of iterations is the same. In some cases, where the target distribution is not sharply peaked, and there are not too many local extreme areas, Rejection-Free can also have excellent performance. Note that if we run the above simulation on a single core, PNS will only take about half of the time used by Rejection-Free, and if we use parallel hardware to apply the above algorithm, Rejection-Free and PNS will take about the same time.

In addition, Rejection-Free is not always better than simple Simulated Annealing. For example, if $\pi(X) \equiv 1$ for all $X \in S$, there will be no rejections. The Simulated Annealing will move to a new state by computing a single probability, while the Rejection-Free will do the same but compute the probabilities for all neighbors. However, when the dimension of the problem is large or the target density is sharply peaked, the PNS will perform much better than Rejection-Free, and Rejection-Free will perform much better than Simulated Annealing.

5.9 Application to 3R3XOR problem

The 3R3XOR problem is a methodology for generating benchmark problem sets for Ising machines devices designed to solve discrete optimization problems cast as Ising models introduced by [41]. The Ising model, named after Ernst Ising, is concerned with the physics of magnetic-driven phase transitions [14], and we have already shown an example for the sampling version of the Ising models

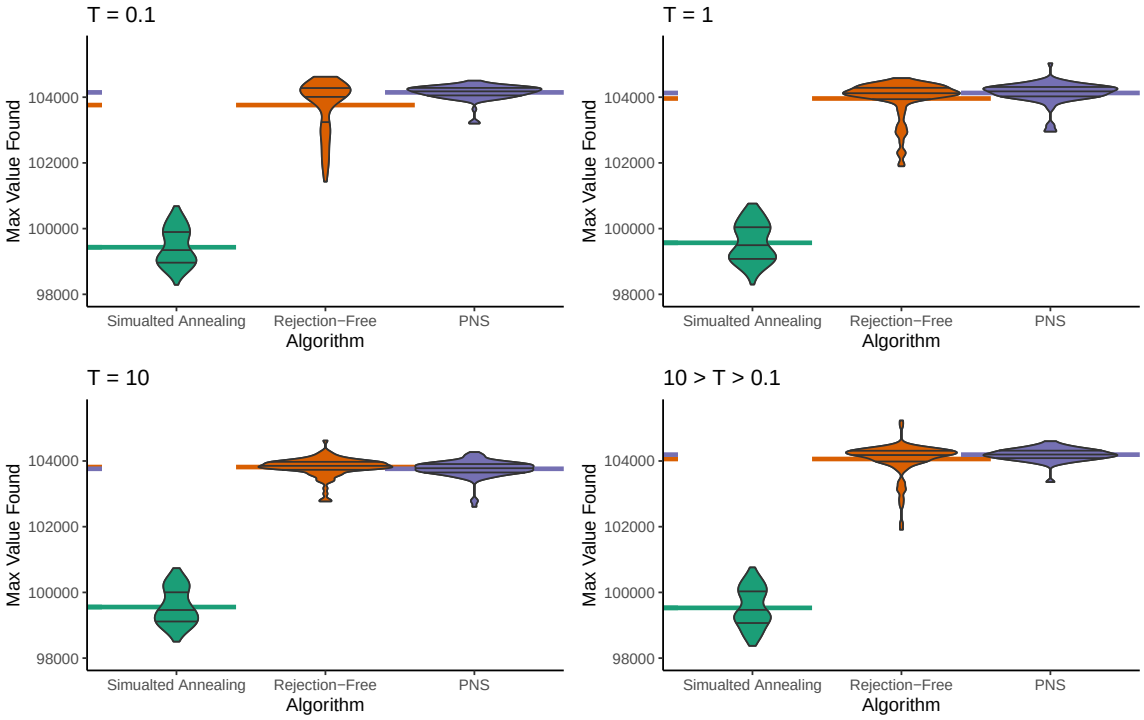

Figure 5.7: Comparison of Simulated Annealing, Rejection-Free, and PNS in terms of the highest target density values found in Knapsack Problem with $W = 100,000$, $N = 1000$, $w_i, v_i \sim \text{Poisson}(1000)$. Four different cooling schedules where $T = 0.1$, 1, and 10 constantly, and T being geometric from 10 to 0.1, are used there. The number of iterations for Simulated Annealing is 1,000,000, while the number for Rejection-Free and PNS is 1000. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

in Section 3.7. Again, the Ising model is defined on a lattice, where a spin $s_i \in \{-1, 1\}$ is located on each lattice site [9]. The optimization question for the Ising model has been widely applied to many scientific problems such as neuroscience [44] and environmental science [53]. Thus, algorithms, even special-purpose programmable devices, designed to solve discrete optimization problems cast as Ising models are popular [41], and our PNS algorithm is one of them.

However, the non-planar Ising model is NP-complete [15]. We cannot find an optimal state from an Ising model in polynomial time. Then, it is hard for us to compare the performance of the heuristic solvers, such as Rejection and PNS, by the time used to find the optimal state from a random Ising model. On the other hand, [41] introduced a tool for benchmarking Ising machines in 2019. In that approach, linear systems of equations are cast as Ising cost functions. The linear systems can be solved quickly, while the corresponding Ising model exhibits the features of NP-hardness [41]. This way, we can construct special Ising models with a unique known optimal state. Then we can use these special Ising models to compare the heuristic solvers' runtimes to find the optimal state.

In this section, we focus on constructing a simplified version of 3-body Ising with N spins from a binary linear system of N equations. The simplified version is defined as follows:

$$H(\{s_j\}) = \sum_{a<b<c} \mathbf{M}_{a,b,c} s_a s_b s_c, \tag{5.4}$$

where $s_i \in \{-1, 1\}$ for $\forall i = 1, 2, \ldots, N$. $\mathbf{M}_{a,b,c}$ is a $N \times N \times N$ matrix where $\mathbf{M}_{a,b,c} = 0$ $\forall a \geq b$, $b \geq c$, or $a \geq c$.

In Hen's (2019) approach, we start by choosing a binary matrix $\{\mathbf{A}_{i,j}\}$ and a binary vector $\{\mathbf{b}_j\}$ to form a modulo 2 linear system of N equations in N variables.

$$\sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \equiv \mathbf{b}_i \mod 2, \text{ for } i = 1, 2, \ldots, N. \tag{5.5}$$

This module 2 linear system of equations can always be solved in polynomial time using Gaussian elimination. In addition, as long as the binary matrix $\{\mathbf{A}_{i,j}\}$ is invertible, the solution (if exists) is unique. Suppose $\{x_1, ..., x_n\}$ are n binary variables. Then for given $\{\mathbf{A}_{i,j}\}$ and $\{\mathbf{b}_j\}$, we define

$$F(\{x_j\}) = \sum_{i=1}^{N} \mathbb{1}\Big(\sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \not\equiv \mathbf{b}_i \mod 2 \Big), \tag{5.6}$$

where $\mathbb{1}$ means indicator function here. Since F is a sum of N indicator functions, then $0 \leq F \leq N$ and the minimum bound is reached when $\{x_j\}$ is the solution to the modulo 2 linear system.

Let $s_j = 1 - 2x_j \in \{-1, 1\}$ for $j = 1, 2, \ldots, N$ be N Ising spins. Then we must have

$$\prod_{j:\mathbf{A}_{i,j}=1} s_j = (-1)^{\mathbf{b}_i} \text{ if and only if } \sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \equiv \mathbf{b}_i \mod 2, \tag{5.7}$$

$\forall i = 1, 2, \ldots, m$. Then

$$\begin{aligned} F &= \sum_{i=1}^{N} \mathbb{1}\Big(\sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \not\equiv \mathbf{b}_i \mod 2 \Big) \\ &= \sum_{i=1}^{N} \mathbb{1}\Big(\prod_{j:\mathbf{A}_{i,j}=1} s_j \neq (-1)^{\mathbf{b}_i} \Big), \text{ since } \prod_{j:\mathbf{A}_{i,j}=1} s_j \text{ and } (-1)^{\mathbf{b}_i} \in \{-1, 1\} \\ &= \frac{1}{2}\Big[\sum_{i=1}^{N} \Big(1 - (-1)^{\mathbf{b}_i} \prod_{j:\mathbf{A}_{i,j}=1} s_j \Big) \Big]. \end{aligned} \tag{5.8}$$

After dropping immaterial constants, we define

$$F_0(\{s_j\}) = \sum_{i=1}^{N} \left[(-1)^{\mathbf{b}_i} \prod_{j:\mathbf{A}_{i,j}=1} s_j \right]. \tag{5.9}$$

Note that $F \geq 0$ and the minimum bound is reached when $\{x_j\}$ is the solution to the modulo 2 linear system. Thus, $F_0 \leq N$, and the maximum bound will be reached when $\{x_j \mid x_j = \frac{1}{2}(1 - s_j)\}$ is the solution to the modulo 2 linear system. In addition, as long as the matrix $\{\mathbf{A}_{i,j}\}$ is invertible, the solution to the equation system must uniquely exist, and then there must exist a single configuration maximize F_0 whose maximum value is exactly N.

Again, the Hamiltonian for simplified 3-body Ising model including only the cubic term to be $H(\{s_j\}) = \sum_{a<b<c} \mathbf{M}_{a,b,c} s_a s_b s_c$. Here, we assume, on each row of binary matrix $\{\mathbf{A}_{i,j}\}$, $\sum_{j=1}^{N} \mathbf{A}_{i,j} = 3$. Then, let $\mathbf{M}_{a,b,c} = (-1)^{\mathbf{b}_i}$ if $\exists i, a < b < c$ such that $\mathbf{A}_{i,a} = \mathbf{A}_{i,b} = \mathbf{A}_{i,c} = 1$, and $\mathbf{M}_{a,b,c} = 0$ otherwise. Then, we have $H(\{s_j\}) = F_0(\{s_j\})$.

Thus, we can construct an Ising model with a unique optimal bound with a known optimal value N as follows:

1. find an invertible binary matrix $\{\mathbf{A}_{i,j}\}$ and a binary vector $\{\mathbf{b}_i\}$, where $\sum_{j=1}^{N} \mathbf{A}_{i,j} = 3$, $\forall i$
2. solve the modulo 2 linear equation system $\sum_{j=1}^{N} \mathbf{A}_{i,j} x_j \equiv \mathbf{b}_i \mod 2$, for $i = 1, 2, \ldots, N$ to make sure the unique solution exists
3. define $\mathbf{M}_{a,b,c}$ be a $N \times N \times N$ matrix where $\mathbf{M}_{a,b,c} = (-1)^{\mathbf{b}_i}$ if $\exists i, a < b < c$ such that $\mathbf{A}_{i,a} = \mathbf{A}_{i,b} = \mathbf{A}_{i,c} = 1$, and $\mathbf{M}_{a,b,c} = 0$ otherwise
4. then we must have a unique optimal solution $s_{\max}$ for $H(s_{\max}) = \max(H(s)) = N$

By constructing the special 3-body $N \times N \times N$ Ising model with a unique optimal solution of maximum bound N, we can examine the performance of the Rejection-Free and PNS algorithms on these special Ising models. Again, uniform proposal distributions are used here, and the neighbors are defined as binary vectors with Hamming distance 1. We random generate the special Ising models with four different sizes $N = 12$, 24, 48 and 96. For each of these four different sizes, we generate 50 different Ising models and record the time used by the algorithms to reach the unique optimal state. The median of these 50 results for both Rejection-Free and PNS algorithms are shown in Figure 5.8. From this figure, Rejection-Free is the worst. 25% PNS performs comparably to 75%, and the 50% PNS performs the best.

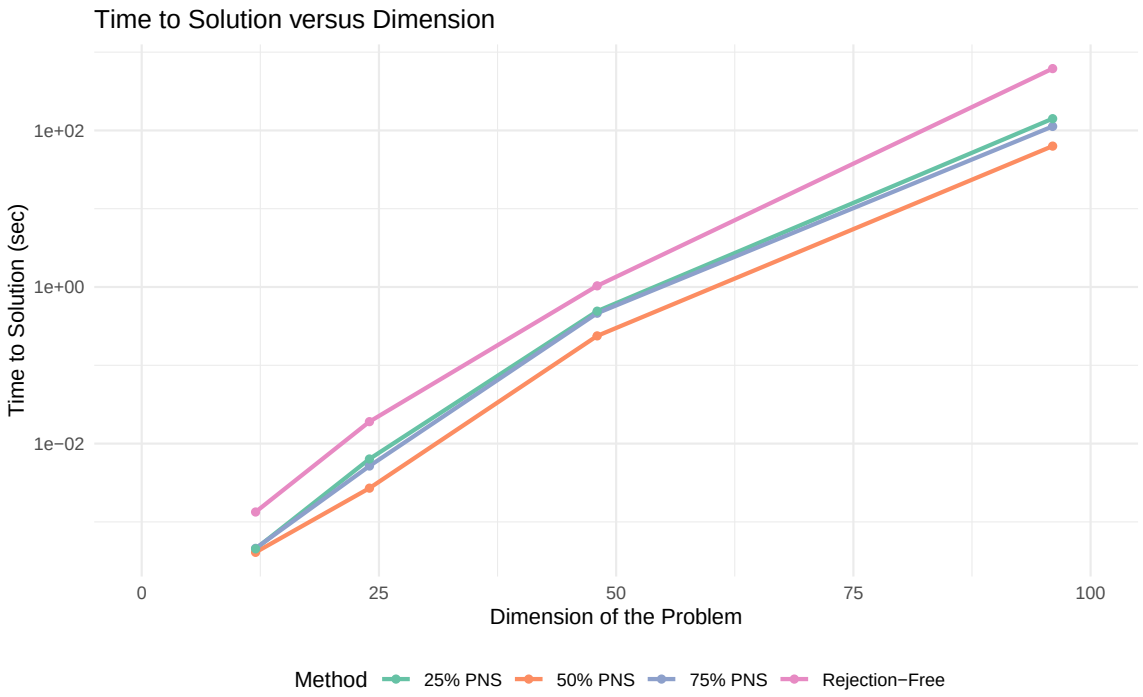

Figure 5.8: Comparison of the minimum value for the time used to find the optimal state by Rejection-Free and PNS with 25%, 50%, and 75% of the neighbors being considered at each step for a random Ising model generated by 3R3XOR. Each dot represents the median of 50 repeated simulations for a given problem size $N = 12$, 24, 48 and 96.

5.10 Application to Continuous State Space

In previous sections, we focused on optimization questions with the discrete state space $\mathcal{S}$ where all states have at most a finite number of neighbors. Meanwhile, Simulated Annealing works for general state space. In addition, Theorem 3.3.6 extended the Rejection-Free for sampling to general state space. Similarly, we can extend the Rejection-Free for optimization to general state space.

Although we have a solid theory base for Rejection-Free in general state space, it is challenging to apply Rejection-Free to those cases. There is a major difficulty involved in the for loop that calculates the transition probability of all neighbors in Algorithm 12. In continuous cases, although numerical integration of all transition probability can be performed, it is unlikely that such tasks may be efficiently divided among specialized hardware with a certain number of parallel processing units. On the other hand, PNS, as described in Algorithm 13, can be applied straightforwardly to continuous cases by choosing the Partial Neighbors Sets $\mathcal{N}_k(X)$ to be finite subsets of all the neighbors $\mathcal{N}(X)$ in Algorithm 13.

We compare the performance of Simulated Annealing with our PNS on a simple example of quadratic programming, which belongs to the category of continuous optimization, as stated below:

$$\begin{aligned} &\arg\max \ x^T Q x \\ &\text{subject to } x_i \geq 0, \forall i = 1, 2, \ldots, N \\ &\sum_{i=1}^{N} x_i = 1, \end{aligned} \tag{5.10}$$

where Q is a given an upper triangular N by N matrix and $x \in \mathbb{R}^N$. For most cases, the quadratic programming is stated by arg min instead of arg max. We use the arg max version here to be consistent with the QUBO question in Section 5.4, and arg max is equivalent to arg min when replacing Q by $-Q$. This quadratic programming question is also NP-hard as long as Q is indefinite [71], where indefinite means matrices that are neither positive semi-definite nor negative semi-definite.

We randomly generate a 200 by 200 upper triangular to be the matrix Q, where the non-zero elements from the 200 by 200 upper triangular matrix Q were generated randomly by $Q_{i,j} \sim \text{Normal}(0, 100^2)$, $\forall i \leq j$. We compare Simulated Annealing and PNS in 100 simulation runs here. We omit Rejection-Free since applying Rejection-Free to continuous cases is quite hard.

The target density value is set to be $\pi(x) = \exp\{x^T Q x\}$, $\forall x$ such that $x_i \in (0, 1)$, $\forall i = 1, 2, \ldots, N$, and $\pi(x) = -\infty$ otherwise. In addition, the proposal distribution $\mathcal{Q}$ and the corresponding neighbor set $\mathcal{N}$ are defined as follows:

1. for state $x = (x_1, x_2, \ldots, x_N)^T \in \mathcal{S}$, choose a random entry x_r for $r \in \{1, 2, \ldots, N\}$;
2. generate a random value $s \sim \text{Normal}(0, 0.1^2)$;
3. let $y_r = x_r + s$ and $y_n = x_n \times \frac{1-x_r}{1-y_r}$, $\forall n \neq r$;
4. if $y_r \notin (0, 1)$, then the corresponding $\pi(y)$ is defined to be $-\infty$; in practice, we just need to generate a new y; also note that, as long as $y_r, x \in (0, 1)$, we must have $y_n \in (0, 1)$ as well;
5. to ensure the reversibility within each Partial Neighbor Set, we also consider $y'_r = x_r - s$ and $y'_n = x_n \times \frac{1-x_r}{1-y'_r}$, $\forall n \neq r$; if $y'_r \notin (0, 1)$, then we can ignore y'.

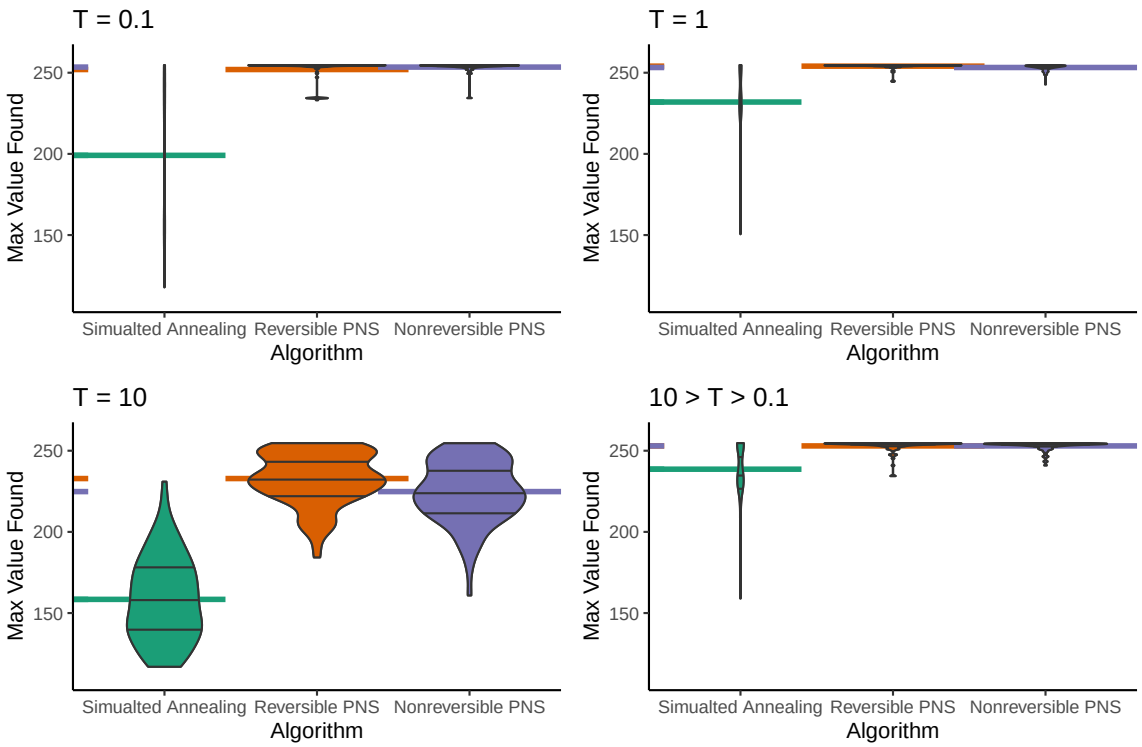

Figure 5.9: Comparison of Simulated Annealing and PNS in terms of the highest (log) target distribution value $\log \pi(x) = x^T Q x$ being found, for a random upper triangular matrix Q and $x \in \mathbb{R}^N$ subject to $x_i \geq 0$, $\forall i = 1, 2, \ldots, N$, and $\sum_{i=1}^{N} x_i = 1$. The non-zero elements are generated by $Q_{i,j} \sim N(0, 100^2)$. Four different cooling schedules where $T(k) = 0.1$, 1 and 10 constantly, and $T(k)$ being geometric from 10 to 0.1 are used here. The number of iterations for Simulated Annealing is $600,000$, and the number of iterations for PNS is $72,000$. The run times for these two algorithms on a single-core implementation are both around 80 seconds. The three black lines inside the violin plots are 25%, 50%, and 75% quantile lines. The colored segments represent the mean values.

With the given steps, we have $\sum_{n=1}^{N} y_n = 1$ as long as $\sum_{n=1}^{N} x_n = 1$. This method is similar to component-wise Simulated Annealing. We find a random component, magnify or minify it, and then modify the rest of the entries accordingly to make the summation remain unchanged. This proposal distribution $\mathcal{Q}$ is therefore systematic. By the above ways to generate neighbors, we can eliminate the constraints that $x_i \geq 0$, $\forall i = 1, 2, \ldots, N$, and $\sum_{i=1}^{N} x_i = 1$, and we only need to focus on $\arg\max x^T Q x$.

For Simulated Annealing, we randomly generate one neighbor by the above-given steps and calculate the transition probability. For PNS, we can generate, for example, 20 random neighbors at each step. In this case, the Partial Neighbor Set $\mathcal{N}_i$ is only a random subset of $\mathcal{N}$ with 20 elements, and thus, the implementation of PNS is simple compared to the Rejection-Free.

The result for the simulation are shown in Figure 5.9. We can see that the PNS performs better than Simulated Annealing in all four different cooling schedules. However, the difference between PNS and Simulated Annealing in this continuous example is not as much as the difference between the algorithms from the discrete QUBO questions. This is because the continuous example is not as sharply peaked as the discrete example from Section 14. After we choose a random entry r, we only need to move a small step around the original value of x_r. On the other hand, we have to

flip between 0 and 1 in the discrete example. Thus, the rejection rate for the Simulated Annealing is lower than the rate from the discrete example, so the performance of these two algorithms gets closer.

In addition, PNS is specially designed for parallel hardware. Again, with a specialized dedicated processor such as DAU, PNS can yield 100x to 10,000x speedups [76]. In addition, this example also shows PNS has more flexibility compared to the Rejection-Free algorithm. Again, Rejection-Free can hardly work for cases with infinite neighbors, while PNS can be easily applied by choosing finite $\mathcal{N}_k$.

Moreover, the number of elements in $\mathcal{N}_k$ needs to be reasonable for PNS to keep its performance. For example, if we used $|\mathcal{N}_k| = 500$, we would calculate too many transition probabilities at each step, and the algorithm would be inefficient. Meanwhile, if we used $|\mathcal{N}_k| = 2$, the number of Partial Neighbor Sets being considered at each step would be too few. As PNS will force the Markov chain to move to one element from the Partial Neighbor Set $\mathcal{N}_k$, it will move to some terrible choices of states when all states in the Partial Neighbor Set $\mathcal{N}_k$ have small target distribution values. In the above simulation, choosing $|\mathcal{N}_k|$ from 10 to 30 won't make a big difference.

5.11 Burn-In by Partial Neighbor Search

In Chapter 4, when we need K samples (original sample size), we have to generate $2K$ states of the original sample size. The first K states are used as the burn-in part. In this way, we started our sampling process from convergence for Rejection-Free and Unbiased PNS. However, Geyer argued that burn-in is not necessary for MCMC in [33]. As an alternative to burn-in, any point that you don't mind having in a sample is a good starting point. His argument suggests that we can start at any point as long as the target density value for this point is large. Geyer claimed that this alternative method is usually better than regular burn-in in [33].

If we accept the above statement, then we can apply optimization algorithms from Chapter 5 before we start sampling. For example, we can cancel the burn-in part of the QUBO question in Section 4.3. Instead, before we start sampling from the target density, we consider optimization algorithms that try to maximize $\pi(x) = \exp\{x^T Q x\}$ for $x \in \{0,1\}^N$ as in 5.4. Then we can start sampling from a state with a large $\pi(x)$ value, although it may not be the optimal one. Simulated Annealing is one such algorithm. In addition, we can also use Optimization Rejection-Free and Optimization PNS from Chapter 5.

However, the starting states obtained by the proposed three optimization algorithms won't converge to the target density. Therefore, people can use these optimization methods to replace the burn-in part only if they believe that the MCMC can start at non-convergence, just like Geyer [33].

However, some people may insist on starting from stationarity. Then we can combine the algorithm for optimization and sampling and try to take advantage of both versions to get a burn-in algorithm. We can apply the optimization algorithm for a certain number of steps K_0, and then we apply the sampling algorithms such as Rejection-Free 12 or Unbiased PNS (Algorithm 7) for K_1 samples.

To check the distribution of the states after a certain number of steps of the mixed algorithm, We generate a certain number of Markov chains by the algorithms, and we record the last state of each chain. As a result, we can get the distribution after burn-in, and we call this distribution to be

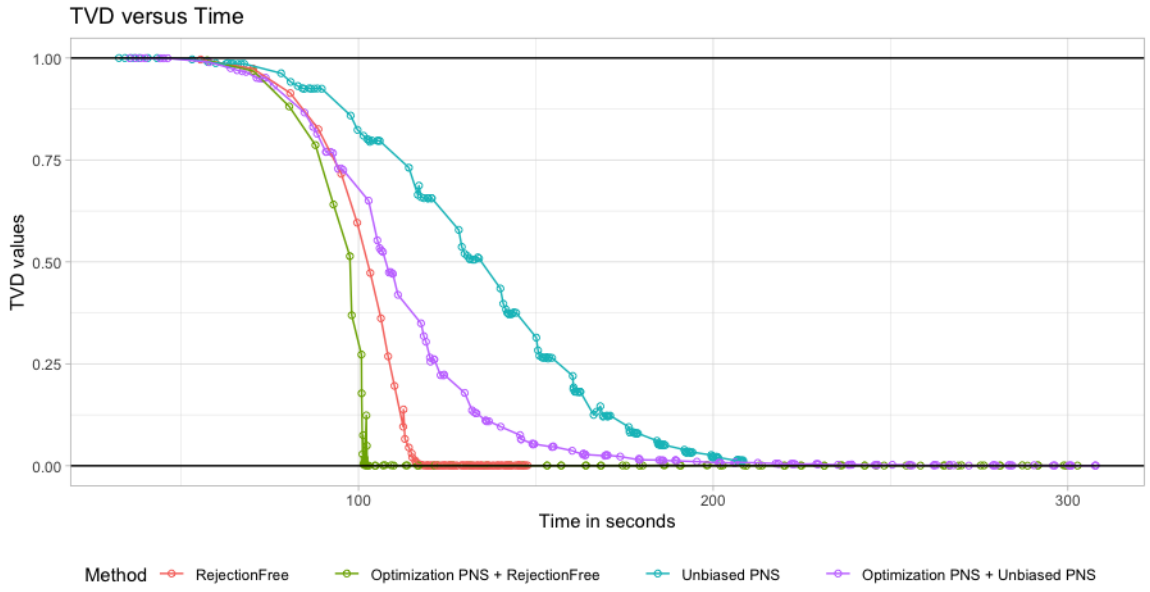

Figure 5.10: Average total variation distance (TVD) between the starting distribution from $100,000$ chains and target density as a function of average time for the chains in seconds for four methods: Rejection-Free, Optimization PNS plus Rejection-Free, Unbiased PNS, and Optimization PNS plus Unbiased PNS. Random upper triangular 16×16 QUBO matrix is generated randomly by $Q_{i,j} \sim N(0, 1^2)$ for upper triangular elements. The original sample sizes for Rejection Free are $K_1 = \{20, 30, 40, 50, \ldots, 1000\}$, and the number of steps for the corresponding Optimization PNS is $K_0 = \lfloor \frac{K_1}{20} \rfloor\}$. The original sample sizes for Unbiased PNS are $K_1 = \{40, 50, 60, \ldots, 1500\}$, and the number of steps the corresponding Optimization PNS is $K_0 = \lfloor \frac{K_1}{40} \rfloor$. Each dot within the plot represents the TVD value between the target distribution π and the distribution of the last state of $100,000$ Markov chains.

the starting distribution for sampling. For example, just like the previous example in Section 4.3, we still consider a 16×16 QUBO question. Every setting is exactly the same as what we have in Section 4.3 except we used $Q_{i,j} \sim \text{Normal}(0, 1^2)$, $\forall i \leq j$. For a standard deviation of 10, we will need much more samples and thus a much longer time to get a small TVD value. since we only use the last states from one Markov chain. We generate one such starting distribution with $100,000$ Markov chains and check the TVD value between the starting distribution and the target density. The number of steps for Optimization PNS K_0 is chosen to be $\lfloor \frac{1}{20} K_1 \rfloor$ ($\lfloor\ \rfloor$ represents the floor function). The number of samples $K_1 = 20, 40, 60, \ldots, 200$. In addition, we also compare Unbiased PNS with Optimization PNS plus Unbiased PNS. Since we believe Unbiased PNS will converge slower than Rejection-Free, so we choose $K_1 = 40, 80, \ldots, 600$, and $K_0 = \lfloor \frac{1}{40} K_1 \rfloor$. The results is shown in Figure 5.10

From Figure 5.10, algorithms with the help of Optimization PNS converge faster with respect to the CPU time. We used a 16×16 QUBO question here. However, we concluded that the higher the dimension is, the more sharply peaked the distribution is, the better the Optimization PNS will be. Optimization PNS performs extremely well in the optimization version of 200×200 QUBO question in Section 5.4. Thus, we can use the Optimization PNS to help burn-in in high dimension or sharply peaked distributions.

Chapter 6

Conclusion

In this thesis, we demonstrate new methods for sampling and optimization questions based on the Markov chain Monte Carlo. By considering all the neighbors at once, Rejection-Free can improve the sampling efficiency over Metropolis-Hastings. Also, two versions of the Partial Neighbor Search algorithms of sampling were introduced to address the problem that Rejection-Free cannot be applied to some specialized hardware. Although the Basic PNS is straightforward, it does not converge in distribution to the target density. Meanwhile, the Unbiased PNS will correctly converge in distribution to the target density with more effort. In addition, the Unbiased PNS can use specialized parallel hardware such as DA [57] to improve sampling efficiency significantly. As compared to a single-core implementation for the QUBO question, the Unbiased PNS performed worse than Rejection-Free. Furthermore, Rejection-Free is infeasible in many continuous cases, while Unbiased PNS can be applied to all continuous cases and works much better than Metropolis. Additionally, for the optimization questions, Optimization Rejection-Free algorithms can prevent the inefficiency of the Simulation Annealing from rejections by considering all neighbors at each step. We have also proposed an Optimization PNS based on the Rejection-Free technique in order to address the issue of the local maximum area. Three sets of discrete examples have been simulated to demonstrate that PNS can produce significant speedups in optimization problems. PNS has also been applied to continuous examples in order to demonstrate its superior flexibility in comparison to Rejection-Free. Additionally, as Optimization PNS outperforms both Simulated Annealing and Optimization Rejection-Free, we can use Optimization PNS to improve burn-in parts before sampling.

There are several directions that can be explored in the future. For the sampling part, what are the optimal choices for the Partial Neighbor Sets and L_0? We compared the sampling efficiency for different combinations of the Partial Neighbor Sets and L_0 for the QUBO question in Section 4.6 and concluded that the choices are highly based on the QUBO matrices themselves and when we get rid of the influence of the matrices, the choice of them does not matter as long as the values are not extreme. However, for other questions, the answers may be different. In addition, we explored the two flips for the QUBO question in Section 4.6.6. We concluded that two flips are useless in our simulations, where the QUBO matrix Q is randomly generated. On the other hand, in some specific cases, it may be helpful. Moreover, we also talked about the irreversible Partial Neighbor Sets, and we concluded that for the Donuts example if we use symmetric proposal distribution, the PNS chain will converge in distribution to the target density, while the asymmetric proposal distribution will

cause the failure of convergence. We can explore the convergence requirements for the non-reversible Partial Neighbor Sets since such sets are more flexible than the reversible ones. Especially for parallel hardware, communication between cores can be hard. Reversible Partial Neighbor Sets do require such communication, and thus non-reversible Partial Neighbor Sets can improve sampling efficiency then.

For optimization questions, the situation is simpler than that of the sampling question. PNS works much better than Rejection-Free and Simulated Annealing on many optimization questions, and in addition, for optimization questions, there is no convergence problem. Thus, the conditions for the PNS convergence Theorem 4.5.3 can be somehow loosened. A very interesting question is how much we can loosen the conditions. For example, we think that Non-reversible Partial Neighbor Sets can be very useful in optimization questions. As long as the Non-reversible PNS is able to find the global maximum state, it can be applied, and we don't need to worry about the convergence properties. In addition, we can also explore the two flips for the optimization question. The information from the QUBO matrix Q can also be used by the proposal distribution to improve the optimization efficiency.

Overall, we have found several algorithms including Rejection-Free, Unbiased PNS, and Optimization PNS that can be used for sampling and optimization, and the use of parallel hardware can make these algorithms even more efficient. We have explored many ways to improve the efficiency of sampling and optimization by choosing different parameters or making small changes to algorithms, and there are still a variety of topics that are worth exploring to improve the efficiency even further.

Appendix A

Notation Used and Their Meanings

1. (a) $\mathcal{N}$ represents the neighbor set for the states; For example $\mathcal{N}(x)$ means the neighbors for state x; see Section 2.3 for more details;
   (b) $\mathbb{N}$ represents the set for natural numbers $\{0, 1, 2, 3, \dots\}$;
   (c) $N(i) = \#\{n \geq 1 : X_n = i\}$ represents the total number of times that the chain hits i (without counting time 0); see Section 2.1.2 for more details;
   (d) N usually represents the dimension of the problem; for example, when we used $N \times N$ QUBO matrix to test the sampling efficiency in Section 4.3 for $N = 16$;
2. (a) $\mathbf{P}$ represents the probability function; for example, if we toss a coin, $\mathbf{P}(\text{head}) = 0.5$;
   (b) $\mathcal{P}$ represents the proportion of a state within a Markov chain; for example, if we have a Markov chain with states $\{A, B, A, C, D\}$, then $\mathcal{P}(A) = 0.4$; see Section 4.1 for more details.
   (c) $\mathbb{P}$ represents the power set; for example, if we have a set $\mathcal{S} = \{A, B, C\}$, then the power set $\mathbb{P}(\mathcal{S}) = \{\emptyset, \{A\}, \{B\}, \{C\}, \{A, B\}, \{B, C\}, \{C, A\}, \{A, B, C\}\}$
3. (a) S: the spin map for the Ising model; see Section 3.7.2 for more details;
   (b) $\mathcal{S}$: the state space for the Markov chains; see Section 2.1 for more details;
4. $\mathbb{E}$ represents the expectation; for example, if $U \sim \text{Uniform}(0, 1)$, then $\mathbb{E}(U) = 0.5$;
5. $\mathbb{R}$ represents the set of real numbers;
6. $\mathbb{1}$ means indicator function; for example, $\mathbb{1}(U < 0.5) = 1$ if and only if $U < 0.5$, and $\mathbb{1}(U < 0.5) = 0$ otherwise;
7. $\setminus$ is the set minus sign; for example, $\{1, 2, 3\} \setminus \{1\} = \{2, 3\}$, and $\{1, 2, 3\} \setminus \{2, 5\} = \{1, 3\}$;
8. (a) $\lfloor a \rfloor$ represents the floor function; for example, $\lfloor 5.1 \rfloor = \lfloor 5.9 \rfloor = 5$
   (b) $\lceil a \rceil$ represents the ceiling function; $\lfloor 5.1 \rfloor = \lfloor 5.9 \rfloor = 6$

Bibliography

[1] B. Albright, "An introduction to simulated annealing", *The College Mathematics Journal*, vol. 38, no. 1, pp. 37–42, 2007.

[2] S. Almasi George and G. Allan, *Highly parallel computing*, 1989.

[3] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An introduction to MCMC for machine learning", *Machine learning*, vol. 50, pp. 5–43, 2003.

[4] C. Andrieu and G. O. Roberts, "The pseudo-marginal approach for efficient Monte Carlo computations", *The Annals of Statistics*, vol. 37, no. 2, pp. 697–725, 2009.

[5] A. A. Barker, "Monte carlo calculations of the radial distribution functions for a proton? electron plasma", *Australian Journal of Physics*, vol. 18, no. 2, pp. 119–134, 1965.

[6] I. Beichl and F. Sullivan, "The Metropolis algorithm", *Computing in Science & Engineering*, vol. 2, no. 1, pp. 65–69, 2000.

[7] D. Bertsimas and J. Tsitsiklis, "Simulated annealing", *Statistical science*, vol. 8, no. 1, pp. 10–15, 1993.

[8] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr, "A survey on metaheuristics for stochastic combinatorial optimization", *Natural Computing*, vol. 8, no. 2, pp. 239–287, 2009.

[9] B. Block and T. Preis, "Computer simulations of the Ising model on graphics processing units", *The European Physical Journal Special Topics*, vol. 210, no. 1, pp. 133–145, 2012.

[10] A. B. Bortz, M. H. Kalos, and J. L. Lebowitz, "A new algorithm for Monte Carlo simulation of Ising spin systems", *Journal of Computational Physics*, vol. 17, no. 1, pp. 10–18, 1975.

[11] R. Bozick, E. Lauff, and J. Wirt, "Education longitudinal study of 2002 (ELS: 2002): A first look at the initial postsecondary experiences of the high school sophomore class of 2002", *National Center for Education Statistics*, 2007.

[12] S. Brooks, A. Gelman, G. Jones, and X. L. Meng, *Handbook of Markov chain Monte Carlo*. CRC press, 2011.

[13] M. Chen, C. Gao, and Z. Ren, "A general decision theory for Huber's ϵ-contamination model", *Electronic Journal of Statistics*, vol. 10, no. 2, pp. 3752–3774, 2016.

[14] B. A. Cipra, "An introduction to the Ising model", *The American Mathematical Monthly*, vol. 94, no. 10, pp. 937–959, 1987.

[15] B. A. Cipra, "The Ising model is NP-complete", *SIAM News*, vol. 33, no. 6, pp. 1–3, 2000.

[16] R. Colwell, "The chip design game at the end of moore's law", in *2013 IEEE Hot Chips 25 Symposium*, IEEE Computer Society, 2013, pp. 1–16.

[17] A. F. Costa and M. Rahim, "Economic design of X charts with variable parameters: The Markov chain approach", *Journal of Applied Statistics*, vol. 28, no. 7, pp. 875–885, 2001.

[18] G. Deligiannidis and A. Lee, "Which ergodic averages have finite asymptotic variance?", *The Annals of Applied Probability*, vol. 28, no. 4, pp. 2309–2334, 2018.

[19] R. Douc and C. P. Robert, "A vanilla Rao-Blackwellization of Metropolis-Hastings algorithms", *The Annals of Statistics*, vol. 39, no. 1, pp. 261–277, 2011.

[20] A. Doucet, M. K. Pitt, G. Deligiannidis, and R. Kohn, "Efficient implementation of Markov chain Monte Carlo when using an unbiased likelihood estimator", *Biometrika*, vol. 102, no. 2, pp. 295–313, 2015.

[21] R. Durrett, *Essentials of stochastic processes*. Springer International Publishing, 1999.

[22] D. J. Earl and M. W. Deem, "Parallel tempering: Theory, applications, and new perspectives", *Physical Chemistry Chemical Physics*, vol. 7, no. 23, pp. 3910–3916, 2005.

[23] R. Eckhardt, "Stan Ulam, John von Neumann", *Los Alamos Science*, p. 131, 1987.

[24] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir", *Information processing letters*, vol. 97, no. 5, pp. 181–185, 2006.

[25] M. Falcioni and M. W. Deem, "A biased monte carlo scheme for zeolite structure solution", *The Journal of chemical physics*, vol. 110, no. 3, pp. 1754–1766, 1999.

[26] C. Feng, *MCMC interactive gallery*, https://chi-feng.github.io/mcmc-demo/app.html?algorithm=RandomWalkMH&target=donut, Last accessed on 2022-07-05, 2021.

[27] C. A. Floudas and P. M. Pardalos, *Encyclopedia of optimization*. Springer Science & Business Media, 2008.

[28] P. A. Gagniuc, *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[29] M. Garey, D. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems", *Theoretical Computer Science*, vol. 1, no. 3, pp. 237–267, 1976.

[30] A. Gelman, W. R. Gilks, and G. O. Roberts, "Weak convergence and optimal scaling of random walk Metropolis algorithms", *The annals of applied probability*, vol. 7, no. 1, pp. 110–120, 1997.

[31] C. J. Geyer *et al.*, "Computing science and statistics: Proceedings of the 23rd symposium on the interface", *American Statistical Association, New York*, vol. 156, 1991.

[32] C. J. Geyer, "Markov chain Monte Carlo maximum likelihood", *Computing Science and Statistics, Proceedings of the 23rd Symposium on the Interface*, pp. 156–163, 1991.

[33] C. J. Geyer, "Introduction to Markov chain Monte Carlo", in *Handbook of Markov Chain Monte Carlo*, CRC Press, 2011, pp. 3–48.

[34] W. R. Gilks, S. Richardson, and D. Spiegelhalter, *Markov chain Monte Carlo in practice*. CRC press, 1995.

[35] F. Glover, "Tabu search—part I", *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[36] F. Glover, "Tabu search—part II", *ORSA Journal on computing*, vol. 2, no. 1, pp. 4–32, 1990.

[37] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using QUBO models", *arXiv preprint: 1811.11538*, 2018.

[38] U. H. Hansmann, "Parallel tempering algorithm for conformational studies of biological molecules", *Chemical Physics Letters*, vol. 281, no. 1-3, pp. 140–150, 1997.

[39] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications", *Biometrika*, vol. 57, pp. 97–109, 1970.

[40] B. Hayes *et al.*, "First links in the Markov chain", *American Scientist*, vol. 101, no. 2, p. 252, 2013.

[41] I. Hen, "Equation planting: A tool for benchmarking Ising machines", *Physical Review Applied*, vol. 12, no. 1, p. 011 003, 2019.

[42] J. L. Hintze and R. D. Nelson, "Violin plots: A box plot-density trace synergism", *The American Statistician*, vol. 52, no. 2, pp. 181–184, 1998.

[43] D. B. Hitchcock, "A history of the Metropolis-Hastings algorithm", *The American Statistician*, vol. 57, no. 4, pp. 254–257, 2003.

[44] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the national academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[45] G. Iliopoulos and S. Malefaki, "Variance reduction of estimators arising from Metropolis-Hastings algorithms", *Statistics and Computing*, vol. 23, no. 5, pp. 577–587, 2013.

[46] M. H. Kalos and P. A. Whitlock, *Monte Carlo methods*. John Wiley & Sons, 2009.

[47] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing", *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[48] G. Kochenberger, J. K. Hao, F. Glover, *et al.*, "The unconstrained binary quadratic programming problem: A survey", *Journal of Combinatorial Optimization*, vol. 28, no. 1, pp. 58–81, 2014.

[49] G. Korniss, M. A. Novotny, and P. A. Rikvold, "Parallelization of a dynamic Monte Carlo algorithm: A partially rejection-free conservative approach", *Journal of Computational Physics*, vol. 153, no. 2, pp. 488–508, 1999.

[50] D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev, "Why the Monte Carlo method is so important today", *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 6, no. 6, pp. 386–392, 2014.

[51] J. S. Liu, F. Liang, and W. H. Wong, "The multiple-try method and local optimization in metropolis sampling", *Journal of the American Statistical Association*, vol. 95, no. 449, pp. 121–134, 2000.

[52] B. D. Lubachevsky, "Efficient parallel simulations of dynamic Ising spin systems", *Journal of Computational Physics*, vol. 75, no. 1, pp. 103–122, 1988.

[53] Y. P. Ma, I. Sudakov, C. Strong, and K. M. Golden, "Ising model for melt ponds on Arctic sea ice", *New Journal of Physics*, vol. 21, no. 6, p. 063 029, 2019.

[54] S. Malefaki and G. Iliopoulos, "On convergence of properly weighted samples to the target distribution", *Journal of Statistical Planning and Inference*, vol. 138, no. 4, pp. 1210–1225, 2008.

[55] A. A. Markov, "Extension of the law of large numbers to dependent quantities", *Izv. Fiz.-Matem. Obsch. Kazan Univ.(2nd Ser)*, vol. 15, no. 1, pp. 135–156, 1906.

[56] L. Martino, "A review of multiple try mcmc algorithms for signal processing", *Digital Signal Processing*, vol. 75, pp. 134–152, 2018.

[57] S. Matsubara, M. Takatsu, T. Miyazawa, *et al.*, "Digital annealer for high-speed solving of combinatorial optimization problems and its applications", in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2020, pp. 667–672.

[58] N. Metropolis, "The beginning", *Los Alamos Science*, vol. 15, pp. 125–130, 1987.

[59] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines", *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[60] N. Metropolis and S. Ulam, "The Monte Carlo method", *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[61] S. P. Meyn and R. L. Tweedie, *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[62] A. G. Nikolaev and S. H. Jacobson, "Simulated annealing", *Handbook of metaheuristics*, pp. 1–39, 2010.

[63] P. H. Peskun, "Optimum monte-carlo sampling using markov chains", *Biometrika*, vol. 60, no. 3, pp. 607–612, 1973.

[64] G. O. Roberts and J. S. Rosenthal, "Geometric ergodicity and hybrid Markov chains.", *Electronic Communications in Probability [electronic only]*, vol. 2, pp. 13–25, 1997.

[65] G. O. Roberts and J. S. Rosenthal, "Two convergence properties of hybrid samplers", *The Annals of Applied Probability*, vol. 8, no. 2, pp. 397–407, 1998.

[66] G. O. Roberts and J. S. Rosenthal, "Optimal scaling for various Metropolis-Hastings algorithms", *Statistical science*, vol. 16, no. 4, pp. 351–367, 2001.

[67] G. O. Roberts and J. S. Rosenthal, "Minimising MCMC variance via diffusion limits, with an application to simulated tempering", *The Annals of Applied Probability*, vol. 24, no. 1, pp. 131–149, 2014.

[68] J. S. Rosenthal, *A first look at stochastic processes*. World Scientific, 2019.

[69] J. S. Rosenthal, A. Dote, K. Dabiri, H. Tamura, S. Chen, and A. Sheikholeslami, "Jump Markov chains and rejection-free Metropolis algorithms", *Computational Statistics*, pp. 1–23, 2021.

[70] R. A. Rutenbar, "Simulated annealing algorithms: An overview", *IEEE Circuits and Devices magazine*, vol. 5, no. 1, pp. 19–26, 1989.

[71] S. Sahni, "Computationally related problems", *SIAM Journal on computing*, vol. 3, no. 4, pp. 262–279, 1974.

[72] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: A survey", *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 127–144, 1975.

[73] A. Schrijver, "On the history of combinatorial optimization (till 1960)", *Handbooks in operations research and management science*, vol. 12, pp. 1–68, 2005.

[74] E. Schuster, "Buffon's needle experiment", *The American Mathematical Monthly*, vol. 81, no. 1, pp. 26–29, 1974.

[75] J. Seco and F. Verhaegen, *Monte Carlo techniques in radiation therapy*. CRC press Boca Raton, FL: 2013.

[76] A. Sheikholeslami, "The power of parallelism in stochastic search for global optimum: Keynote paper", in *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference*, IEEE, 2021, pp. 36–42.

[77] A. C. Sodan, J. Machina, A. Deshmeh, K. Macnaughton, and B. Esbaugh, "Parallelism via multithreaded and multicore CPUs", *Computer*, vol. 43, no. 3, pp. 24–32, 2010.

[78] Y. Sugita and Y. Okamoto, "Replica-exchange molecular dynamics method for protein folding", *Chemical physics letters*, vol. 314, no. 1-2, pp. 141–151, 1999.

[79] R. H. Swendsen and J. S. Wang, "Replica Monte Carlo simulation of spin-glasses", *Physical review letters*, vol. 57, no. 21, p. 2607, 1986.

[80] L. Tierney, "A note on metropolis-hastings kernels for general state spaces", *Annals of applied probability*, pp. 1–9, 1998.

[81] M. H. Weik, "The ENIAC story", *Ordnance*, vol. 45, no. 244, pp. 571–575, 1961.

[82] T. J. Wu, Y. C. Hsieh, and L. A. Li, "Statistical measures of DNA sequence dissimilarity under Markov chain models of base composition", *Biometrics*, vol. 57, no. 2, pp. 441–448, 2001.